DESIGN OF MULTICAST SWITCHES FOR SANS

APPROVED BY SUPERVISING COMMITTEE:

____

Dr. Rajendra V. Boppana, Supervising Professor

____

Dr. Turgay Korkmaz

____

Dr. Weining Zhang

Accepted: ____

Dean of Graduate Studies

DESIGN OF MULTICAST SWITCHES FOR SANS

by

RAJESH BOPPANA, B.Tech

THESIS
Presented to the Graduate Faculty of
The University of Texas at San Antonio
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
May 2003.

Acknowledgements

I would like to thank Dr. Rajendra V. Boppana, my supervisor for guiding me throughout this thesis. His problem solving and analysis set an example to me and influenced my approach to the problem. I also learned a lot from his simple and creative presentation.

I express my gratitude to the Computer Science Division and its staff for extending excellent cooperation at all the times.

May 2003

DESIGN OF MULTICAST SWITCHES FOR SANS

Rajesh Boppana, M.S.
The University of Texas at San Antonio, 2003

Supervising Professor: Dr. Rajendra V. Boppana

As desktop workstations become more and more powerful and demand for network bandwidth increases, switch-based networks of workstations are becoming a popular choice for high performance computing as well as other applications such as web servers, multimedia servers, data mining, graphics and visualization. The success of such architectures depends on the capabilities of the switches used in the network. With the increased use of multicast applications, it becomes necessary for the switches to be able to handle multicast traffic efficiently in addition to the unicast traffic.

In this thesis we describe an input-output buffered switch based on a banyan type Ω network that uses a simple round robin type scheduling. The switch is based on multiple copies of Ω network as switch fabric and employs a hardware cell scheduling mechanism. There is no restriction on the fanout of the multicast cell. The hardware complexity of the switch is $O(NlogN)$ for an $N \times N$ switch and compares favourably with $O(N^2)$ complexity of a crossbar based switch design. Our architecture is better than the other multistagenetworks in terms of switching elements used as well as maximum sustainable throughput.

Using extensive simulations, we show that our switches provide as much as 50% higher throughput than known crossbar based switches. We also simulated switched storage area networks using the proposed multicast switch as the building block and demonstrate 50% or more throughput can be achieved compared to crossbar based switched networks.

Contents

List of Figures

Chapter 1

Introduction

The last few years have seen explosive growth of network applications such as teleconferencing, multiplayer games and streaming continuous media by Internet radio. With the Ethernet technology pacing towards tens of gigabits per second line rates and current CMOS technologies providing lower transmission delays, the future of distributed computing environment seems to be provided by clusters of off-the-shelf computers interconnected by high speed switches (gigabit LANs). The widespread proliferation of cheap and powerful handheld devices with less memory is paving the way for architectures with separate computing and storage devices. Storage area networks (SANs) based on this trend have already gained popularity. SANs differ from local area networks (LANs) in terms of delay constraints and traffic patterns. In a typical SAN, read requests from clients are unicast (one-to-one) messages, writes by clients are correlated multicast (one-to-any) messages and the response to the read requests are correlated unicast messages.

Workstation clusters may be designed by interconnecting computing and storage devices through broadcast-oriented shared medium or point-to-point switches 1.1. In contrast to networks that use a broadcast physical medium such as the ethernet, a switched LAN/SAN offers (a) aggregate network bandwidth that is much larger than the throughput of single link, (b) the ability to add throughput incrementally by adding extra switches and links to match network-load requirements, and (c) a more flexible approach to the availability using multiple redundant paths between hosts. Brocade, Qlogic, JNI, Inrange, SUN, IBM are some of the SAN switch vendors. Current technology provides throughput of up to 2Gbps per port and up to 32Gbps with inter switch links and switch sizes vary from 8 to 224 ports.

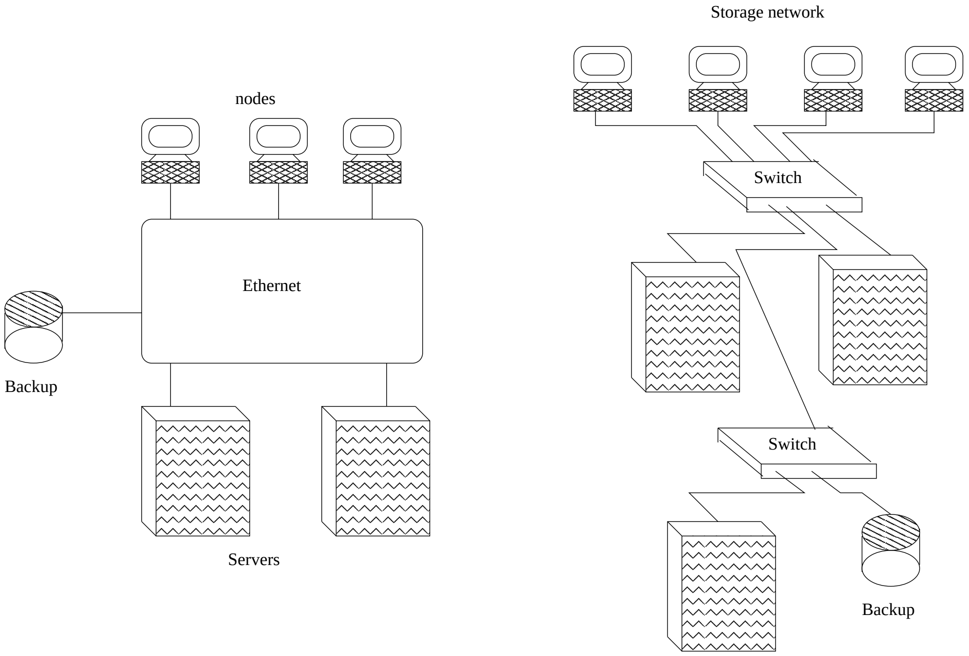


Figure 1.1: An example Ehternet and Storage networking architecture.

Low-cost, high-speed switches are critical for the proliferation of such systems and applications. These switches in addition to handling unicast traffic should also handle multicast traffic efficiently. However, the current switched network designs are based on switches that can not handle multicast traffic. Multicast traffic was handled at the software level by the host adapters [42, 45]. The problem of routing multicast cells is more complex than routing unicast cells. Since multicast cells from different inputs could request same outputs, the conflicts for the outputs increase. Hence, in addition to determining the input ports to deliver cells we also need to decide upon the number of destinations of the cell each input port should deliver simultaneously so that lower cell latencies and high switch utilization are achieved. A number of different architectures have been proposed for multicast switches.

A key component of these switch designs is the switch fabric or data path used to move cells from input to output ports. Earlier designs are based on time or space division multiplexing switch fabrics. In designs based on time division multiplexing switch fabrics, the inputs share the switch fabric which could be a bus, in a round-robin fashion for a fixed amount of time. In shared memory architectures the inputs and outputs write and read cells to a common memory. Shared medium (either the bus or memory) should operate N times the line rate for an $N \times N$ switch. So the medium becomes a bottleneck as the size of the switch increases or the line rate increases. Space division fabrics are based on crossbar or multistage interconnection networks. For crossbar based designs [11, 18, 13] the cost complexity measured in terms of crosspoints increases at $O(N^2)$ for an $N \times N$ switch. To minimize the cost of switch fabric researchers looked into designs based on multistage interconnection networks such as the banyan network [1, 2, 3]. The banyan networks are self routing in nature and have lower hardware complexity. However, these switch fabrics can not support all possible permutations of input-to-output connection patterns due to their low hardware complexity. So they introduce additional complexity to the cell scheduling problem, which is the need to schedule cells that do not conflict for paths through switch fabric as well as output ports.

In this thesis, we focus on designing multicast switches that can handle multicast traffic efficiently without compromising the unicast traffic. Our design uses a multistage network, specifically an Omega network, as the switch fabric, with input and output buffers and hardware based cell selection [10]. The switch has the cost complexity of $O(N \log N)$ for an $N \times N$ switch. We determine the cells that would not have conflicts in the switch fabric by the hardware selection method and deliver them to their respective outputs. The switch has buffers at the inputs as well as the outputs but not inside the switching elements unlike some of the earlier designs that use multistage networks [1, 9]. This design handles both unicast and multicast traffic and there is no limitaion on the fanout of the multicast cell.

We also look into the performance of the switches interconnected to form a high-speed storage area network. We use the proposed switch as a building block for the switched networks. We simulate two network topologies, one in which a larger switch is simulated by interconnecting smaller switches and the other is a general network formed by interconnecting the switches randomly. Our emphasis is on the performance evaluation of our proposed switch fabric with respect to crossbar based designs in the context of switched networks.

1.1 Organization

The remainder of the thesis is organized as follows. We summarize the existing multicast switch designs in Chapter 2. We describe our switch architecture and the scheduling algorithms in Chapter 3. We describe the simulator used in Chapter 4 and present the simulation results in Chapter 5. Chapter 6 talks about the switched networks and simulation results of two network topologies and Chapter 7 concludes the thesis.

Chapter 2

Background

In this chapter we summarize the existing multicast switch designs, outlining our proposed design. First, we describe the terminology used.

1. $N \times N$ switch: A generic switch has input and/or output buffers to hold the cells, a swich fabric, the data-path to move cells from the inputs to the outputs, and a scheduler to arbitrate and allocate paths through the switch fabric. A switch with N input and N output lines is referred to as $N \times N$ switch.

2. slot time: Time is divided into slots. In the absence of contention or queuing, a cell may be sent completely from inputs to outputs of the switch fabric in a slot time.

3. cell/packet: The cell or packet contains the message or data that is to be routed by the switch from the input lines to the output lines. It could be a unicast cell, in which case it has one destination to be delivered to or a multicast cell, which has more than one destinations. In this thesis we consider cells of fixed size.

4. line rate: Line rate is the speed at which the input or output lines connected to the switch, transfer the data (cells).

5. conflicts: Two or more cells (unicast or multicast) from different inputs could request the same outputs. These are called output conflicts. Some cells which do not have output conflicts could have conflicts for the data paths within the switch fabric. Such switch fabrics are called blocking switch fabrics and switches based on such switch fabrics are blocking switches.

6. scheduling: The process of deciding the cells that could be sent through the switch fabric by each of the input ports so that high throughput and low latencies are achieved is scheduling.

Location of the cell buffers, topology of the switch fabric and the mechanism to resolve the contention for output ports or paths through the switch fabrics are the critical aspects of the switch design that influence the cost and performance.

The main design choices for a switch fabric can be classified as time-division multiplexing and space-division multiplexing. In time-division switch fabrics, the fabric, which is often a bus or memory, is shared among the inputs for a fixed amount of time in a round robin fashion. For example in shared memory designs each input writes one after another consecutively to the shared memory and in shared medium architectures each input sends its data onto the common medium consecutively in turns. Hence the inputs and outputs need to operate at N times the link speed for an $N \times N$ switch. Although such designs are attractive in terms of simplicity of design and performance achieved, they are expensive to implement and are not scalable.

Most of the research for the past few years has been on space-division multiplexing as they provide more bandwidth within the switch. These switch designs are based on crossbar, multistage interconnection networks or fully interconnected networks. Input buffering in which the cells are queued at the input ports is widely used in crossbar based designs [11, 18, 13]. In crossbar switches with input buffering, broadcasting an input cell to multiple output ports is straightforward. The cell could be delivered to all its destinations by setting up the paths to required outputs. When two or more multicast cells compete for a common set of outputs it becomes necessary to allow some of the destinations of a multicast cell to be delivered during one slot time and the remaining destinations to be reachable in subsequent slot times by keeping a copy of the cell in the input queue until it reaches all of its destinations to achieve high throughput [32]. This is called fanout or cell splitting. Weight Based Algorithm (WBA) on crossbar with input buffering proposed by Balaji et al. [11] is known to achieve high throughput.

However, the cost of crossbars in terms of crosspoints increases at $O(N^2)$. In order to reduce the high cost of crossbars, many researchers have explored various architectures [1, 2, 9] based on multistage interconnection networks like banyan networks [33] and delta networks [34]. However, there is not enough bandwidth inside the switch fabric to support all posssible input-to-output connection patterns. So even when several cells request distinct outputs, these networks may not be able to route all the cells simultaneously. Hence, complex scheduling is required to resolve the conflicts for data paths in addition to the conflicts for outputs.

Sort-Banyan networks are proposed to overcome the blocking nature of banyan networks [36] which have a sorting network like Batcher's bitonic sorting network [35] to reorder the cells in increasing order of destinations. The cells so ordered can be routed without path conflicts to the outputs by Banyan networks. These designs can be classified as copy-route networks. They make multiple physical copies of the cell in the switch fabric by using copy networks [1, 2, 9]. Hence, in effect, they route $k * N$ cells through the fabric rather than N cells, where k is a function of the multiplicity of physical copies made. This increases the size of the switch fabric and also the scheduling complexity by a factor of at least k. Arbitrating the conflicts among these replicated cells requires additional paths in the network which is obtained by using extra stages of switching elements or by recirculating them within the fabric. Another possibility is to buffer the excess or undeliverable cells, internally at the switch elements.

Buffers can be placed at the input, output or within the switch fabric or a combination of these three locations. Input buffering is commonly used in the designs based on crossbars. But, cells at the head of the queue having output conflicts do not allow other cells behind in the queue, which do not have output conflicts to pass through the switch fabric. This is called head-of-line blocking and restricts the performance of such designs to 58.6% for uniformly distributed unicast traffic [14]. Various techniques like multi-queue buffers [17], virtual output queuing [26], elaborate cell scheduling policies [13, 18, 22] have been proposed to overcome this problem. With output queuing, it is assumed that all cells at the inputs are delivered to output queues by the switch fabric. Output queuing gives better throughput and delays, but the switch fabric should be capable of delivering multiple cells per slot time to the outputs. Also, either the output buffers must operate at higher speeds than the line rate or there should be multiple buffers at each output. In both cases, the throughput and scalability are limited, either by the speedup factor or by the number of buffers. The Knockout switch [15] and Gauss switch [25] are examples of designs employing output queuing. Buffers can also be placed inside the switching elements in a multi stage interconnect switch fabric [1, 16] or inside the crosspoints in a crossbar [29, 30, 31]. Internal buffers introduce random delays within the switch fabric, causing undesirable cell delay variation and are expensive to implement [5].

To summarize, crossbar based designs do not make multiple physical copies of the cell in the switch fabric. Only one copy of the cell resides at the input buffer of the switch until its delivered to all its destinations. This keeps the design of the switch fabric simple, limiting the design complexity to scheduing cells for each slot time. This modular approach makes the switch adaptable to very high line rates. A major drawback of crossbar based designs is the cost which makes them impractical for medium to large scale sized switches. Another problem is cell-splitting is complex and promotes head-of-line blocking, especially for correlated traffic. We try to simplify the designs based on multistage interconnection networks by taking advantage of pre-schedulng the cells as in crossbar based designs, thereby lowering hardware complexity of the switches.

2.1 Scheduling Policies For Crossbars

In this section we present two well known scheduling policies on crossbars.The WBA proposed by Balaji et. al. [11] for scheduling multicast cells and Parallel iterative matching algorithm (PIM) proposed by Thomas E. Anderson [13] for unicast cell scheduling. These hardware cell selection methods facilitate faster cell selection over software approaches and the switch is scalable to higher line rates. In these algorithms, a special hardware based on a fully connected bipartite topology is used for scheduling. A network is bipartite if it can be divided into two subsets such that all the interconnections in the network have one end in one subset and the other end in the other subset.

Weight Based Algorithm: Weight Based Algorithm (WBA)[11] algorithm works by assigning weights to input cells based on their age and fan-out, where age is the number of cell slot times an input cell has to wait at the input before being transmitted and fan-out is the number of destination ports to which the cell is to be delivered. Inputs send their weights to output ports. Among all the inputs competing for a particular output, the output port chooses the heaviest input port. Incase of multiple requests with same highest weight, the outout chooses an input randomly. Based on their previous experience, they found that to achieve fairness a positive weight has to be given to the age and a negative weight to the fan-out (the older the heavier and the larger the lighter). If f is the weight assigned to fan-out and a is the weight assigned to age, then no cell waits at the input port for more than $M + \frac{fN}{a} - 1$ cell times. Where M is the number of input ports and N is the number of output ports. Mathematically,

$$w = a \times age - f \times fanout$$

where, w is the weight of the cell, age is the number of slot times the cell is at the head of its queue and $fanout$ is the number of destinations of the cell.

Figure 2.1 shows the working of WBA on a 4x4 crossbar switch. A weight of 1 is given to the age and a weight of 2 is given to the fan-out, that is, $a = 1, f = 2$. Initially port 4 gets a chance to transmit

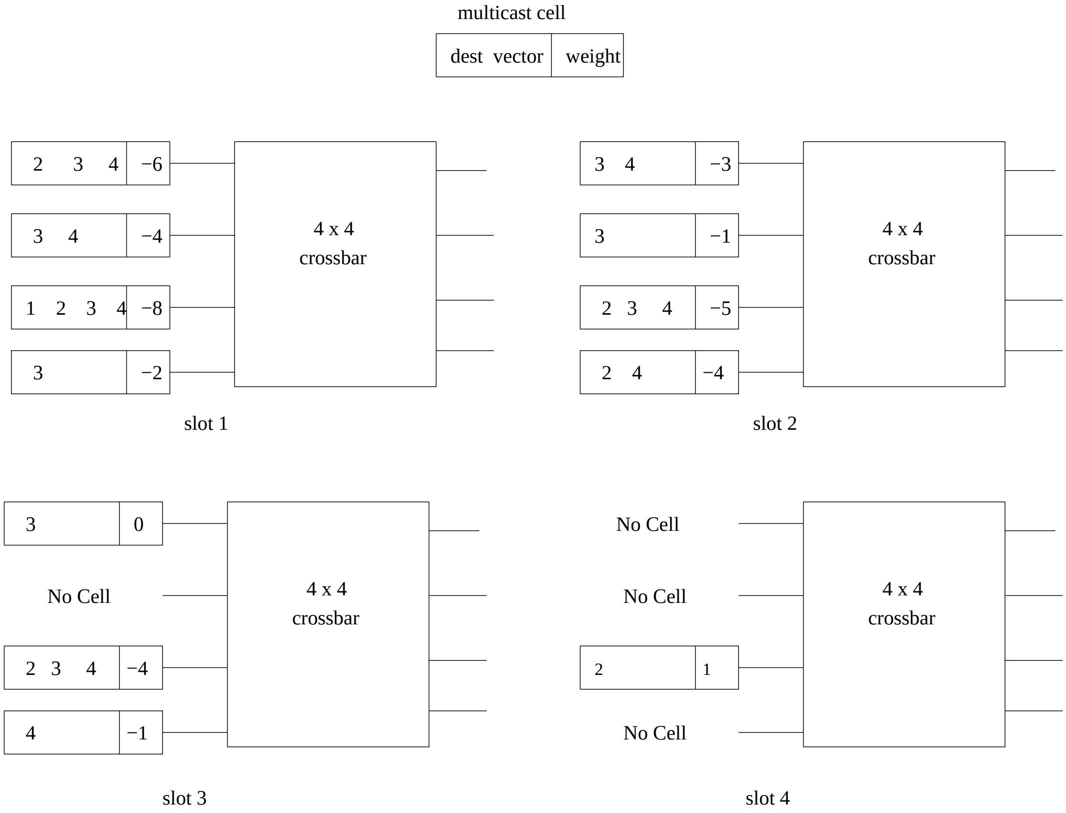


Figure 2.1: Example of WBA on a 4x4 Crossbar Switch. The rectangular blocks indicate multicast cells at the inputs.

its cell as it has the maximum weight -2 among all the competing inputs. In the next slot time it injects new cell from its input queue. In slot 1 node 3 could reach only one destination 1. The cell is split as two cells, one with destination vector $<$1$>$ and the other with destination vector $<$2,3,4$>$. The first cell is transmitted and the second cell remains in the queue and competes with the other cells in the next slot time.

Parallel Iterative Matching algorithm: Parallel Iterative Matching algorithm for scheduling unicast cells uses parallelism, randomness and iteration to find a conflict free pairing of outputs quickly. The following three steps are iterated to find a matching pair. Initially all inputs and outputs are unmatched.

1. Each unmatched input sends a request to every output for which it has a buffered cell. This notifies an output of all its potential partners.

2. If an unmatched output receives any requests, it chooses one randomly. The output notifies each input whether its request was granted.

3. If an input receives any grants, it chooses one and notifies the output of its acceptance.

Each of these steps occur independently and in parallel at each input/output port. After a fixed number of iterations, the result of the matching is used to setup the crossbar for the next time slot.

Chapter 3

Ω switch

In this chapter, we describe the Ω switch, which will be used as a building block for storage area networks. Boppana and Raghavendra presented the Ω switch for unicast traffic [10]. Gunuganti enhanced it to handle mulitcast traffic [47]. We have revised the design and extended it for use in switched networks.

3.1 Switch Design

Our design attempts to simplify the design complexity of switch fabric and output ports at the cost of more complex cell scheduling. We use a combination of input and output buffers and multiple copies of the Omega network (Figure 3.1) as the switch fabric. Figure 3.2 shows the block diagram of Ω switch.

The cells arriving from the input lines are buffered at the input port buffers. Once a cell is in the input buffer, it is guaranteed to reach all of its destinations (switch outputs). Hence there is no cell loss within the switch fabric or output buffers. Cells are lost only when they arrive at the switch and are rejected by the input ports. This happens when the switch is saturated. Based upon the header of the cell, the destinations of the cell are determined by table lookup and a routing tag is assigned for routing through the switch fabric. For now, we may consider routing tag to be an N bit vector with each bit denoting an output port. A cell is destined to an output port if its corresponding bit in the vector is set.

Routing tags could be precomputed or prepared as a cell is placed in an input buffer or when a virtual circuit between sender and destination is set up. We consider only data routing here, not the signaling aspects.

We select cells in such a way that there are no path conflicts within the switch fabric. Our cell selection scheme is based on the hardware cell selection mechanism proposed by Boppana and Raghavendra [10]. As the scheduling mechanism is hardware based the switch can operate at very high speeds. Because cells are carefully selected to eliminate path conflicts there is no need for buffering at any of the switching elements inside the switch fabric. We believe that, by moving all the necessary buffering away from the data path of the switch, we can simplify the switch fabric design.

The input scheduler determines the inputs and the cells they need to send through the data network. Contention for paths within the switch fabric and outputs affect the performance of the switch. When crossbars are used as switch fabrics, there is no contention for data paths through the switch fabrics. It is a severe problem when blocking switch fabrics like Omega network are used. On the other hand using crossbars could be expensive. To alleviate this, we use multiple copies say, k, of the Omega network in the switch fabric. So conflicts for a path in the switch fabric can be handled by sending contending cells through different copies of the network. This increases the bandwidth inside the switch and achieves high throughput. Since we have k copies of the data network, each output could receive up to k cells during a single slot time. Hence the output buffers need to operate at k times the line speed. We can restrict the maximum number of cells sent to an output port in a single slot time to some l, where $l \leq k$. When k is small, it is simpler to have $l = k$. In each slot time, one cell is sent out of the output port. Excess cells are buffered in the output queues. This would achieve high switch utilization since any output that did not receive a cell due to blocking nature of the switch fabric could send out a cell as long as its buffer is not empty. This is particularly useful for correlated traffic in which the demand for an output tends to be bursty. Although, we use multiple copies of Omega network we limit the complexity or speedup requirements of output buffers by controlling the output port contention through

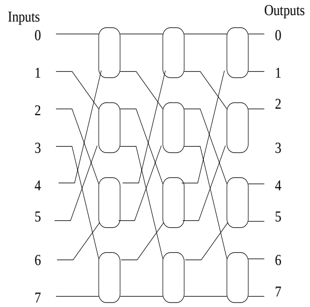


Figure 3.1: An Omega network.

cell scheduling so that the switch is scalable.

3.1.1 Scheduling Cells

It is known that multistage interconnection network based switches need elaborate cell selection techniques to achieve high throughput. Software based cell selection schemes do not work well for high line rates or large switches. We use the idea of hardware specific selection technique proposed by Boppana and Raghavendra [10]. In this method we use a copy of the underlying network (Omega network itself) as the control network to aid the selection of cells. However this control copy of the network only needs to route the routing tags of the cells for scheduling purposes. The routing tag has at most N bits for an $N \times N$ switch. The control copy does not route the data packets. In each slot time, several rounds of selection are performed. The cells that can be sent in a slot time are selected by the scheduler in the previous slot time. Thus scheduling and moving data through the switch fabric are pipelined to avoid increasing the slot time to accomodate scheduling time.

Mulitcast Scheduling

Round-Robin scheduling: The simplest way to schedule multicast cells is to allow one or more input ports to send their multicast cells in a round-robin fashion during each slot time. Since our proposed switch has k data networks, its intuitive to select k input ports to route their multicast cells through the

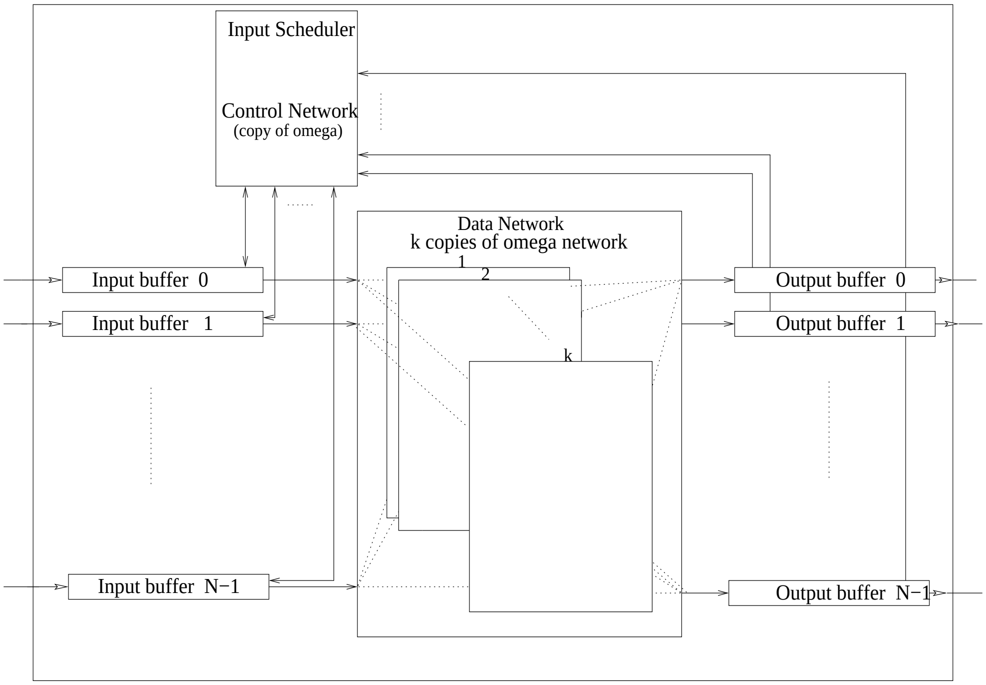


Figure 3.2: Switch architecture. The switch constitutes of input and output queues, control network and data network. The control network is made up of a Omega network that only needs to be bit routable and acts as input scheduler. The data network consists of k copies Omega networks for routing the cells through the switch. (k is a small value like 2 or 3).

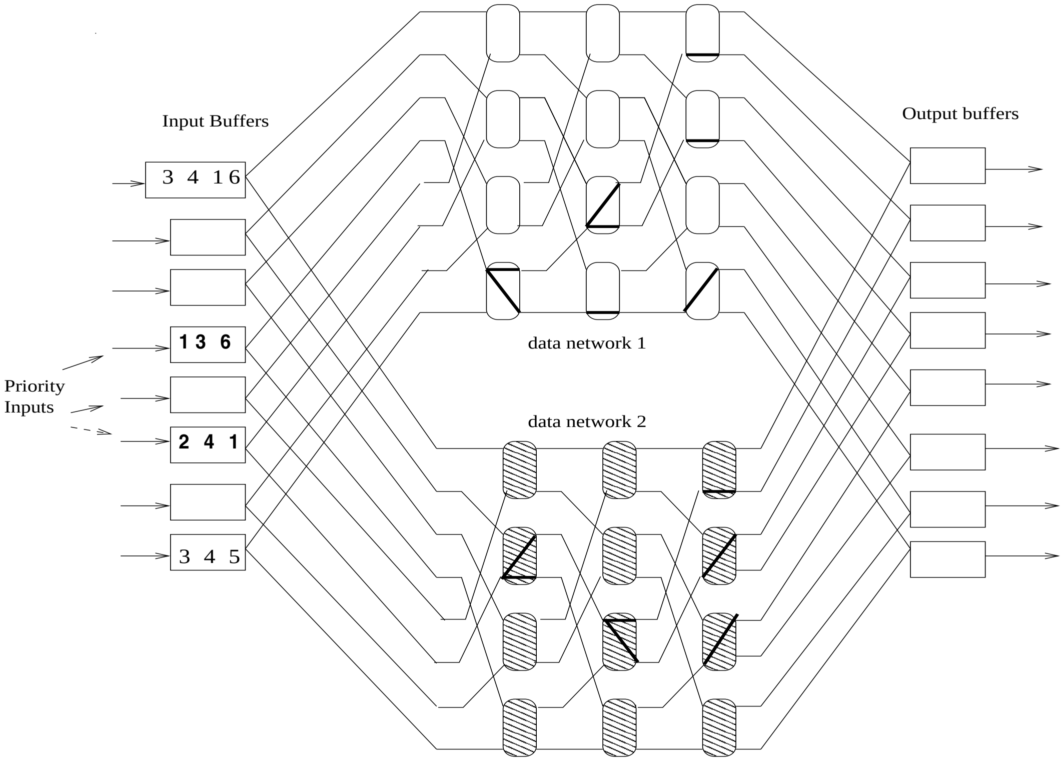


Figure 3.3: Example of multicast cell selection. The diagram shows the selected input ports and their paths through the corresponding data networks. Blocked lines reperesent paths established through the data networks.

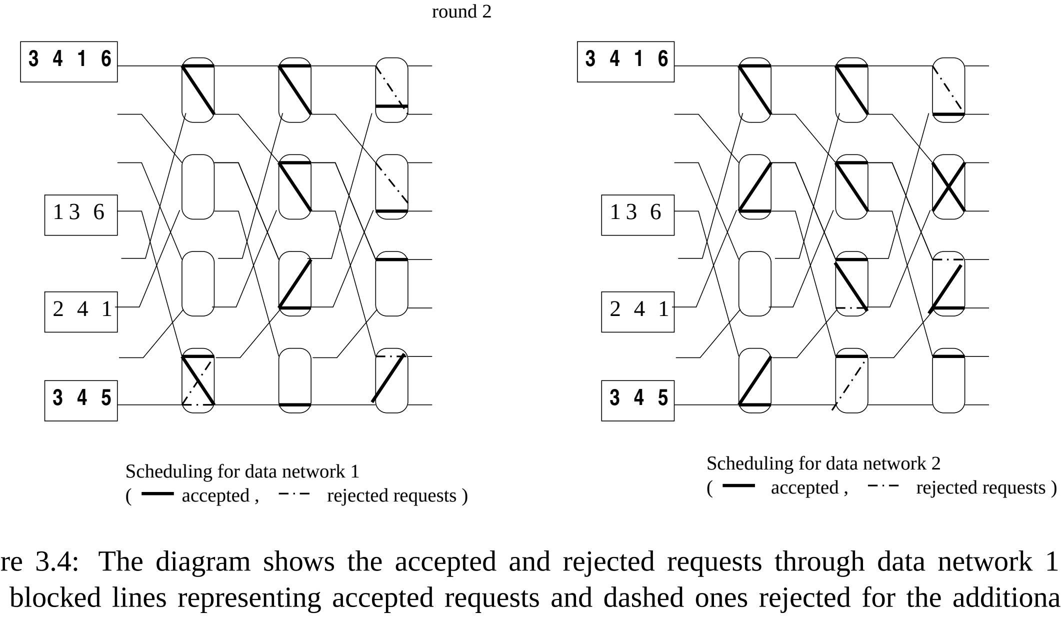

round 2
3 4 1 6
1 3 6
2 4 1
3 4 5
Scheduling for data network 1
(accepted , rejected requests)
3 4 1 6
1 3 6
2 4 1
3 4 5
Scheduling for data network 2
(accepted , rejected requests)

Figure 3.4: The diagram shows the accepted and rejected requests through data network 1 and 2, with blocked lines representing accepted requests and dashed ones rejected for the additional round of scheduling.

k data networks (one through each copy).

Hence, the simplest scheduling policy is a sort of prioritized round robin algorithm in which during each slot a particular input port has the priority and can send its multicast cell unrestricted through a copy of the data network. The next input port (selected in a circular fashion) that has a multicast cell can send its multicast cell unrestricted through another copy of the network and so on. If a prioritized input port doesn't have a multicast cell the next input port (in a circular fashion) gets a chance. The priority to the input ports during a slot time is given in a round-robin manner.

Figure 3.3 shows an example of multicast scheduling. The first stage shows the simplest scheduling scheme where input ports 3 and 4 have priority. Since input port 4 does not have any cell to send, the priority is passed on to the next input port which is indicated by a dashed arrow. This policy basically routes k multicast cells through the switch during each slot time. So switch utilization with simple round-robin scheme is

$$\frac{k \times f}{N}$$

, where k is number of copies of daa network, f is the fanout of the multicast traffic and N is the size of the switch. This scheme does not work well if f is small and N is large.

Scheduling additional cells: To improve the performance, we attempt to schedule more than k multicast cells during each slot time. That is, some data networks send two or more multicast cells without path conflicts. This can be achieved by performing a round of selection for additional multicast cells that could go through each data network. In this scheduling policy, after determining the k input ports that would send their multicast cells through the k data networks, using the round-robin scheduling, we let the remaining input ports send their output requests (routing tags) through the control network and determine if any other multicast cells could be scheduled through each of the data networks without conflicting with the internal route of the earlier selected multicast cell. Hence there is one round of

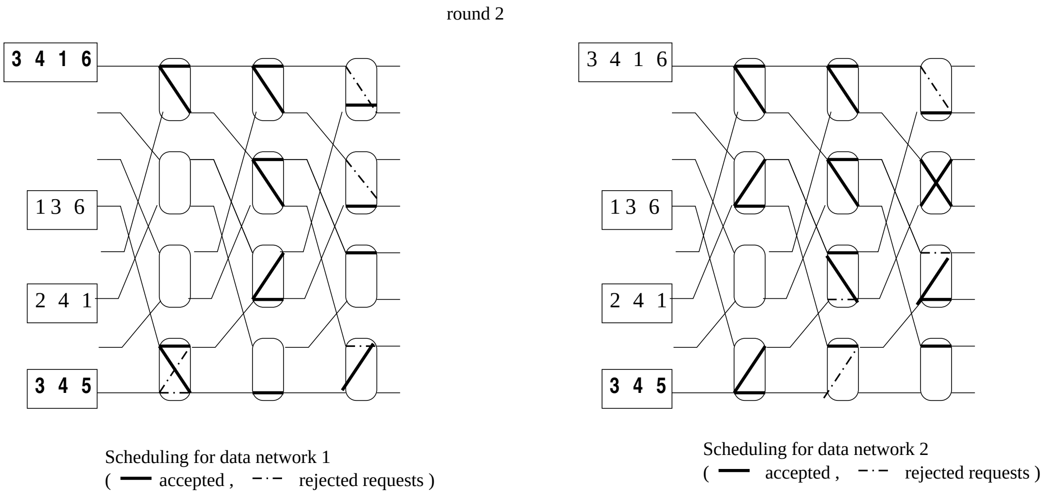

round 2
3 4 1 6
1 3 6
2 4 1
3 4 5
Scheduling for data network 1
(accepted , rejected requests)
3 4 1 6
1 3 6
2 4 1
3 4 5
Scheduling for data network 2
(accepted , rejected requests)

Figure 3.5: Example of multicast cell selection with cell splitting. The diagrams show the accepted and rejected requests through data network 1 and 2, from left to right,respectively, with blocked lines representing accepted requests and dashed ones rejected for the additional scheduling involving splitting.

selection for each of the k copies. It is noteworthy that this additional scheduling improves the switch utilization by using additional data paths. Each input request is either satisfied or rejected in entirety. That is there is no cell splitting.

Figure 3.4 shows forwarding of additional multicast cells through the two copies of the data network. In this example, additional cells could not be scheduled due to conflicts with paths allocated to cells already selected by the round-robin scheme. If the fanout of the multicast cells is high, the probability of finding a cell which uses paths that do not conflict with those of an already selected multicast cell (using round-robin scheme) decreases and this scheduling does not achieve any improvement over the simple round-robin policy. So for multicast traffic with larger fanout, it becomes necessary to split the cell to achieve high switch utilization. We therefore consider fanout splitting which is also used in the crossbar designs discussed in Chapter 2.

Scheduling additional cells with fanout splitting: In this policy after determining the k input ports that would send their multicast cells through the k data networks, the remaining input ports send their routing tags through the control network to determine if any other multicast cells could be scheduled through each of the data networks without conflicting with the internal routes of earlier selected multicast cells. But in this case if a multicast cell is able to obtain paths to reach some but not all of its destinations, we split the destination list and let a copy of the multicast cell go to the available outputs while retaining a copy with the remaining destinations at the input. Once an input port obtains paths for one or more destinations for its multicast cell, it will not compete for paths in the other copies of the data network. When a cell is plit the cell with reduced destination list will be treated as a normal multicast cell for scheduling in later slot times.

Figure 3.5 shows an example of the cell scheduling with cell splitting. The cell from input port 0 with destiantion set 3, 4, 1 and 6 is able to reach the outputs 1 and 3 through first copy of data network. As the cell from input port 0 is able to reach some of its destinations through the first copy it is not considered for scheduling through the second copy of the network. However, it was considered in Figure 3.4 as splitting is not allowed there.

For mixed traffic patterns where both multicast and unicast traffic are present, first, multicast cells are scheduled by one of the three policies and the remaining unused paths in the data networks are used for unicast cells. Unicast scheduling is described in the next Section.

Unicast Scheduling

The scheduling of unicast cells is described in [10]. During each cell slot time we determine the inputs and the cells that they send in the following time slot. Multiple rounds of selection are performed over the control network for achieving high throughput. In the first round, all inputs send their requests (the routing tags of the cells at the head of their queues). The round concludes by notifying the inputs whose requests have been routed successfully through the network. In further rounds, those inputs whose requests were not granted will compete with a different cell (that is the next cell in the queue with a destination different from those that were rejected earlier) for the remaining outputs and internal

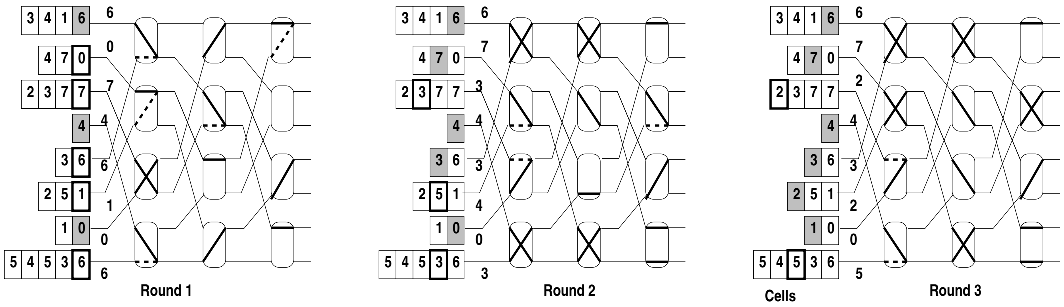


Figure 3.6: Example of unicast cell selection. The rectangular blocks indicate the cells (with numbers indicating their destinations) queued at switch inputs. The numbers at the inputs of the first stage of switching elements indicate the destinations of the cells that inputs wish to send. Shaded cells indicate winning inputs and the cells selected; thick-lined cells indicate inputs with rejected requests. In a switching element, dashed lines indicate rejected connections and solid lines accepted connections. Inputs with rejected connections select new outputs for future rounds. Inputs with accepted connections use the same outputs for later rounds.

switch paths, while the inputs that have succeeded send the same cell again. Conflicts at the switching elements are broken randomly, however, a request that has been successful in earlier rounds (that is the winning cell) always wins in the case of conflict. After few such rounds we know the inputs and the cells that they need to route through the network.

Figure 3.6 gives an example of unicast cell selection. Initially, each input requests for the output specified by its first cell in the queue. In the example, inputs 0, 3, and 6 (when counted from top starting with 0), win in the first round and have paths to their destinations 6, 4, and 0, respectively, established. These paths are not disturbed in further rounds (2 and 3). In round 2 inputs 1, 2, 3, 5 and 7 choose next cell in their queues whose destinations are not 6, 4, or 0 (that is those outputs to which paths have been established in earlier rounds) and those destinations that were tried in earlier rounds (for example, input 1 will select a cell whose destination is not 0, as it was tried in earlier rounds and hence will not succeed in further rounds either). In round 2 two more inputs win and in round 3 one more input wins.

3.1.2 Output Queuing

We use output buffers to store cells to accomodate multiple cells arriving at an output in a slot time. So the output queues could could overflow and result in high cell loss. To prevent this, a back pressure mechanism is used. If the output queue size exceeds a threshold it accepts only one cell during each slot time until the number of cells at the output is less than the threshold. This is achieved by not granting an output with too many cells to more than one cell during the scheduling phase.

Queue length	number of cells accepted by outputs
$<$ threshold	upto k multicast cells
$>$ threshold	only one multicast cell
0	upto k multicast cells or 1 unicast cell

Although output queuing is used, atmost one unicast cell is scheduled to an output port in each slot time. Moreover only those unicast cells are scheduled whose destination output ports have empty buffers (i.e., no outstanding cells to be drained out to the output lines). This policy makes sure that in the presence of both multicast and unicast traffic, multicast traffic does not experience further delays due to unicast traffic. However, the relative priorities of unicast and multicast traffic can be easily changed by modifying the thresholds.

3.1.3 Design Issues

Switch Fabric

The switch fabric described has one control network and k copies of data network. With such an architecture the total time to transmit cells from input port to output port is equal to the sum of delays of scheduling the cells for the k copies over the control network (which is k times the delay of a control network) and the delay due to transmitting cells from the inputs to the outputs through the k data networks simultaneously.

Hence the time taken for transmitting the cells across the switch fabric is $k * d$ where d is the delay of transmitting N bits through the control network and selecting multicast cells. Let D be the delay in transmitting the cells through the data network. Then the proposed scheduling can be achieved in one slot time if $kd \leq D$.

Estimation of d

In an Ω switch fabric with 2 x 2 switching elements, a cell can reach any of the outputs through a switching element in stage 1 and only one half of the outputs through a switching element in stage 2 and one eighth of the outputs through those in stage 3 and so on. More specifically a cell can reach the first half of the outputs through the upper outputs of the switching elements in stage 1 and the second half of the outputs for the switch through the lower output port of a switching element. We therefore need to send only one half the bits of the bit map vector to the next stage. In other words switching elements in stage 2 need only half the bits to make their scheduling decision. Similarly switching elements in stage 3 need only one-fourth the bits to make a routing decision and so on. Let us assume that the control network is implemented using bit-serial lines. If t is time taken to transmit one bit across the switching element, the delay incurred for a round of scheduling is

$$N \times t + \frac{N}{2} \times t + \frac{N}{4} \times t + ... + 2 = 2(Nt - 1).$$

This is particularly attractive for large switches as the hardware required to implement switching decisions are bit-wise functional units (bit comparators) and can be very fast.

Multicast addressing

Until now we have assumed the addressing of multicast cells as a bit map vector of N bits for an $N \times N$ switch with one bit corresponding to each output port. In this case each switching element would keep a bit map vector for each of its output links indicating all the destinations reachable through it. So when a cell comes in, the routing tag is compared to the switching element's bit map vector to determine if it is to be sent out through that output link of the switching element. The complexity of this addressing scheme in bits per routing tag is $O(N)$. Although this scheme works well for switches of small size, it may not be viable for large switches with N ≥ 512.

To reduce the number of bits in the bit map vector of large switches, we can group the outputs and assign a group number. The routing tag would now consist of the group number and the bit map vector to identify the destinations with in that group. The number of bits used to address is sum of the number of outputs in each group and number of bits required to identify the groups (which is log(no. of groups)). For example for 256 x 256 switch divided into 16 groups we need 4 bits to address each group and 16 bits to identify each output in a group which requires 20 bits in total.

Alternatively, we could use the header information of the incoming cell like the VCI and VPI of an ATM cell and by table look-up, set the corresponding switching elements to route the cell. But this scheme does not give the flexibility to split the cells as in the bit-map-vector method. It also increases the complexity of input scheduler as well as the switch fabric as it has to send control signals to all the switching elements in the fabric.

Multicast addressing is a major issue by itself and much work needs to be done in this area. However, any good scheme can be easily incorporated into our design with little or no changes.

Chapter 4

Simulator

In this chapter we describe the simulator used to analyze the proposed designs. The simulator is written in java programming language.

4.1 Program Structure

The program is a time driven model and takes in the number of ports, number of switching elements, unicast traffic load, multicast traffic load, fanout of multicast cell, size of correlation train, size of input and output queues and the type of scheduling policy to be used as arguments from the user. The basic structure of the program is shown in 4.1. It has various classes like Omega.java, Inport.java, Outport.java, Switchelmnt.java to simulate the effect of the scheduler, the input ports , output ports and the switching element of the switch, respectively.

Omega.java: This class simulates the effect of the switch fabric and the scheduler. It has the main method and hence is the starting point of the simulation. It initializes all the necessary data structures and calls the respective methods of all the other classes acting as a controller. It reads the topology information about the switching elements in the fabric from an input file. This file contains information regarding the interconnections among the switching elements and the output ports one could reach through each of these switching elements' outputs. The input ports, switching elements and the output

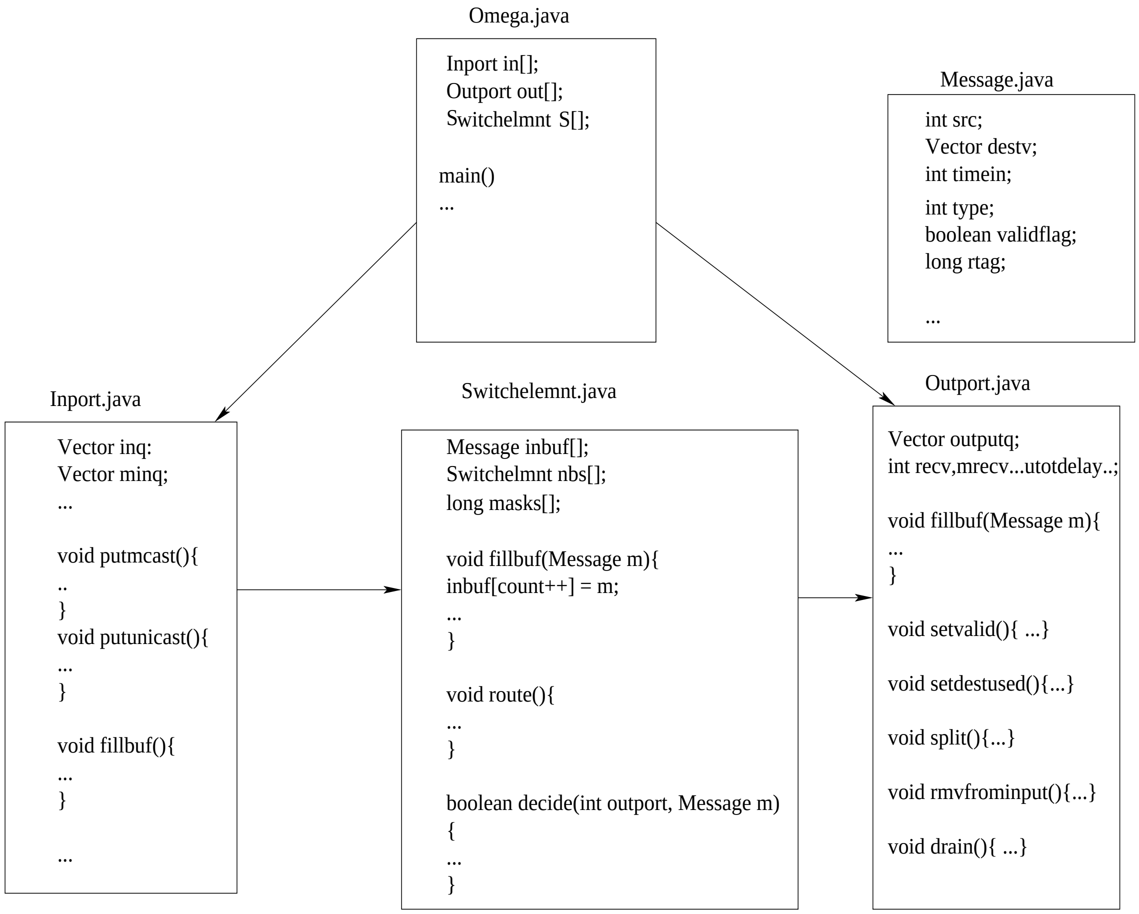


Figure 4.1: Program structure

ports are numbered as shown in 3.1.

The input file for an 8×8 Ω is shown in fig. 4.2. The second line of the input file gives an ordered list of the switching elements to which input ports 0, 1, 2, 3, 4, 5, 6 and 7 are connected. The remaining rows list the ids of the neighboring switching elements and their masks. For example line 4 indicates that the upper output port of switching element 0 is connected to switching element 4 and its lower output is connected to switching element 5. The next 2 columns are the masks of switching element 0. A bit 1 in the mask field indicates a reachable output port. The most significant bit indicates output port 7 and least significant bit output port 0. They signify that through the upper output of the switching element 0, a cell can reach output ports 0, 1, 2 and 3 and through the lower output a cell can reach output

```
 (8x8)    neighbours to input ports:
0 1 2 3 0 1 2 3
sweid nbr0 nbr1      upper mask   lower mask
0  4 5    00001111   11110000
1  6 7    00001111   11110000
2  4 5    00001111   11110000
3  6 7    00001111   11110000
4  8 9    00000011   00001100
5  10 11  00110000   11000000
6  8 9    00000011   00001100
7  10 11  00110000   11000000
8  0 1    00000001   00000010
9  2 3    00000100   00001000
10 4 5    00010000   00100000
11 6 7    01000000   10000000
```

Figure 4.2: Input file for an $8 \times 8 \Omega$ switch.

ports numbered 4, 5, 6 and 7. These values are stored in the switching element class as *masks[]*, with *mask[0]* for the upper output and *mask[1]* for the lower output.

Inport.java: This class models the function of the input ports. The fields, *inq* and *minq* of type vector simulate the effect of an input queue for unicast and multicast cells respectively. The field, *nbr* is a reference to its neighbouring switching element. The methods, *putunicast(),putmcast()* send an unicast or a multicast cell respectively to its neighbouring switching element and *fillbuff()* injects a new unicast or multicast cell into the switch by placing it in the appropriate queue (inq or minq).

Outport.java: This class models the output ports of the switch. The field, *outputq* of type vector simulates the effect of the output queue The fields *recv, mrecv and mcellrecv, utotdelay* and *mtotdelay* keep count of unicast and multicast cells received and their delays. The method, *drain()* removes one cell from the output queue every slot time. The output port receives a cell from the neighbouring switching element through the method *fillbuf(message m)* which is then put into the field, *outputq*.

Switchelmnt.java: This class models the switching element of the switch and is the heart of the switch in routing the cell through the fabric. The method, *fillbuf(message m)* receives cells from its neighbouring switching elements. When a cell comes in, the method *boolean decide(int outport, Message m)* determines if a message *m* should be sent to outport of the switching element by performing a bitwise and of the routing tag of the cell (rtag) and the output mask of the switching element (*masks[0] or masks[1]*). And the method *route()* using this method *decide()* determines which cell and to which output ports of the switching element the cell is to be routed depending on the scheduling policy used. Conflicts between the cells could occur when two multicast cells request both the outputs or when one multicast cell requests one of the outputs and another multicast cell requests both the outputs of the switching element. In such cases one of the cell is randomly dropped and in conflicts between a unicast and multicast cell, the unicast cell is dropped. Several random number streams with different seeds are used to improve randomness. When the scheduling policy used is non fanout splitting a multicast cell either goes to all or none of its requested output ports of the switching element. Whereas in fanout splitting, a cell would be split such that it could go to any number of its requested output ports of the switching element.

Message.java: This class defines the structure of the unicast or multicast cell. It has fields to keep track of the source, the time the cell entered the switch, the routing tag, the destinations it has to go etc. We consider the routing tag of the cell as an N bit vector. It is represented in this class as a long type variable named *rtag*. Each bit in the routing tag represents an output port. The corresponding bits of the *rtag* are set to 1 if a multicast cell has destinations to particular output ports. The bits from least significant bit to the most significant bit correspond to output ports numbered 0 to N-1 respectively. For an example if a multicast cell has 2 and 4 as its destinations then its routing tag would be *00010100* for an 8×8 port switch.

Table.java: This class is for statistical purposes that has methods to calculate the mean and variance. It is used in recording the statistics of each batch of the simulation like delays, utilization, load.

The program used to simulate the crossbar also has similar structure as described above. However, it doesn't have the Switchelmnt.java class and also doesn't need to read any topology information from an input file. The simulation of the WBA design has Wba.java, Inport.java, Outport.java to simulate the effect of the scheduler, the input port and the output port. It aslo has other helper classes like Message.java, Table.java discussed earlier.

4.2 Simulation of large Switches

Since the data type long in java is represented as 64 bits we can only simulate 32×32 Ω by the above method. In order to simulate large switches we modify the program as follows.

We create another class named Dlong which has a field named v of type vector. The idea is to store 32 bits of routing tag as Long type object in the vector v. Suppose we wanted to simulate 64×64 Ω we will store the first 32 bits (which correspond to lower numbered output ports) of the routing tag as a Long type object in the first position of the vector v, v.elementAt(0) and the next 32 bits in the second position of the vector v, v.elementAt(1). Hence for an $N \times N$ switch there are ($\lceil \frac{N}{32} \rceil$) elements in the vector v. The bits in the higher indexed elements of the vector represent the higher numbered output ports.

We also define the necessary operations for this data type like bitwise and and bitwise exclusive or through the methods boolean and(Dlong d) and void xor(Dlong d) respectively.

The data files for large switches are generated by an another program. The generated data file is the same as the input file described earlier. However, the 2 masks are chopped into 32 bit chunks as explained above and represented as long type binary numbers consecutively. For example a sample line for a 64 port switch would look like this. Though in the actual file the masks are binary numbers we show them here as hexadecimal numbers for convenience.

```
0  32 33     0x0000000000000000  0xffffffffffffffff
   0xffffffffffffffff  0x0000000000000000
```

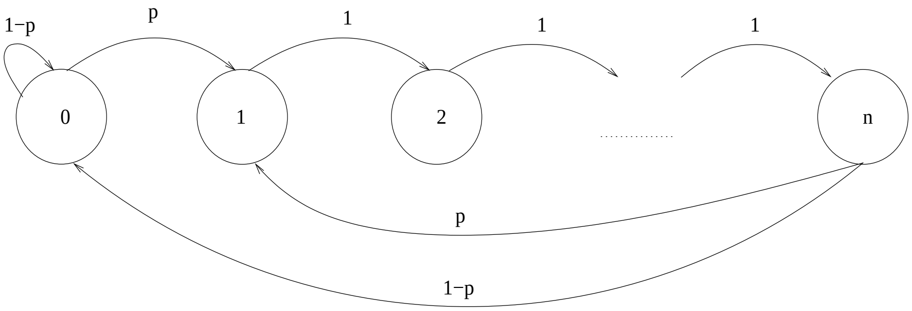


Figure 4.3: N state markov process depicting the state of the input port

4.3 Simulation of Traffic Patterns

The multicast traffic can be simulated as uncorrelated or correlated arrivals. For this purpose, the injection of cells at inputs is simulated by an arrival process.

Uncorrelated Arrivals: At the beginning of each cell time, a cell arrives at each input with probability independent of the arrival of the previous cell time.

Correlated Arrivals: Cells are generated using an n-state Markov process with n consecutive busy states and one idle state (see Figure 4.3). When the arrival process is in busy state a cell is injected into the queue with probability 1 and process goes to the next state. The destinations of a cell are selected randomly in state 1 and used repeatedly in states 2 to n.When in the idle state a transition to busy state occurs with probability p or remains in the same idle state otherwise.

$$p = \frac{cellrate}{(cellrate+(1-cellrate)\times n)},$$

where cell rate is the average number of cells per slot time per input port.

$$cellrate = unicastload + (\frac{multicastload}{fanout}),$$

where unicast load and multicast load are specified by the user as a fraction in the range [0 , 1]. In state

1 a cell is determined as unicast or multicast type with probability

$$q = \frac{unicastload}{unicastload + multicastload}$$

and $1 - q$, respectively. All the multicast cells in a correlated train have the same set of destinations. That is the multicast cells generated in the n consecutive busy states have same destinations.

In light of self-similarity of the magnitude of traffic transmitted on local and wide area networks [48], [49], correlated arrivals model the network characteristics more accurately than uncorrelated arrivals.

Multicast fanout For both types of traffic, simulations were carried out with arriving multicast cells having a constant fanout and variable fanout.

Constant fanout In this kind of traffic the fanout of all the multicast cells is fixed and is specified by the user as an input argument at the beginning of the simulation. We consider constant fanout multicast traffic because in a typical network fanout of multicast cells are ususally small. For example in a storage area network multicast traffic are due to write requests to a few fixed number of servers and hence this kind of traffic will facilitate realistic analysis of the switches for SAN traffic.

Variable fanout In this type of traffic the number of destinations of arriving multicast cells is uniformly distributed in the range of [1 , N] for an N port switch. This type of distribution is known as Bernoulli distribution. The average fanout of the cells in this distribution is $\frac{N+1}{2}$. We use this variable fanout distribution to serve as a basis of comparison with other works in the literature.

Chapter 5

Performance Analysis

5.1 Simulations

We evaluated the performance of the proposed Ω switch design using the simulator described in Chapter 4. We also simulated WBA on crossbar to serve as baseline in our comparisons. For mixed traffic patterns we used WBA for multicast scheduling and PIM for unicast scheduling on crossbar based switches.

We simulated a 8×8, 16×16 and 64×64 Ω- based and crossbar-based switches with separate buffers for unicast and multicast traffic. While each input queue of crossbar can buffer up to 256 multicast cells and 256 unicast cells, Ω can buffer 128 unicast cells and 128 multicast cells at the input and 128 cells at the output. So the total buffer space used was the same for both designs. The control network of the Ω switch used two rounds for multicast (one for each copy) and eight rounds for unicast (four rounds for each copy) per slot. A weight of 1 for the age and -2 for fanout was used for WBA simulations.

Each simulation was run for 100,000 cycles as one batch with a warmup of 50,000 cycles until the mean value of the unicast and multicast latencies of all the batches fell within the 95% confidence interval. We use the following notations for all the plots. Omega, Omega2 and Omega-split denote, respectively, the simple round-robin, round-robin with one round of scheduling and Omega2 with fanout splitting. WBA denotes the weight based algorithm [11] over crossbar based switch fabric.

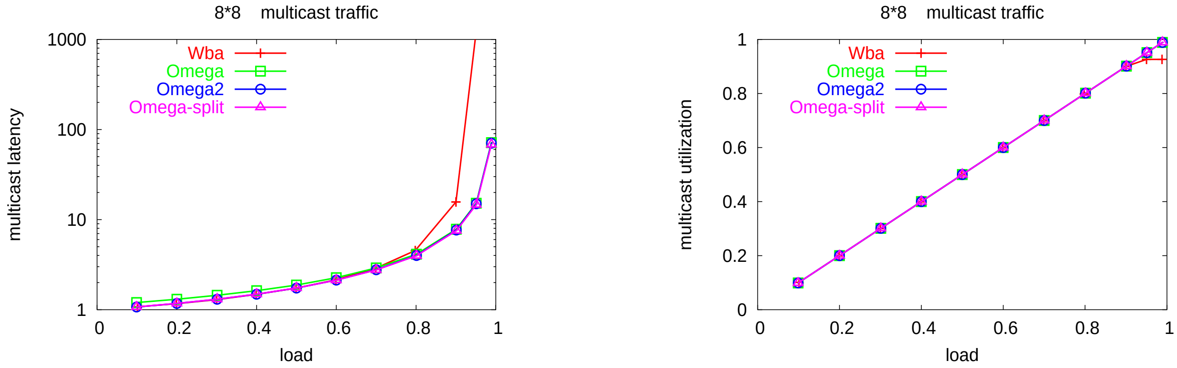


Figure 5.1: Performance of 8×8 switches for uncorrelated multicast traffic with variable fanout. We use the following abbreviations for all plots. Omega : The round robin policy which schedules two multicast cells per slot time over Ω based switch fabric. Omega2 :The round robin scheme and an additional round of selection for multicast cells over Ω based switch fabric. split : The round robin scheme and fanout splitting over Ω based switch fabric. WBA: The WBA multicast scheduling policy over crossbar based switches.

5.1.1 Multicast Traffic Simulations

In this section we present the simulations of multicast traffic. This type of traffic simulates the suitability of the switch for multicasts.

Multicasts with variable fanout

We present the simulation of multicast traffic with Bernoulli distribution of destinations here. Figure 5.1 shows the delay and utilization of various designs for uncorrelated arrivals for an 8×8 switch. We see that Omega, Omega2, and Omega-split achieve nearly 100% utilization and WBA achieves about 95% utilization. Also, the latency curves for Omega, Omega2 and Omega-split are similar and slightly lower than that of WBA.

Figure 5.2 shows the delay and utilization of various designs for correlated arrivals on an 8×8 switch. The input queued WBA design achieved only 78% utilization due to prolonged and increased contention for output ports because of correlated arrivals. However, Omega, Omega2, and Omega-split achieve about 95% utilization because of their abiltity to buffer cells at the output. Also, the latency

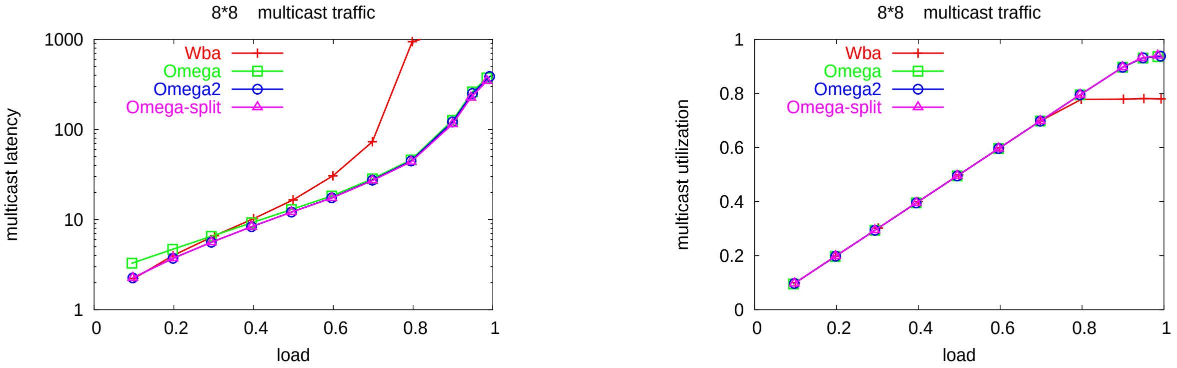


Figure 5.2: Performance of 8×8 switches for correlated multicast traffic with variable fanout.

curves for Omega, Omega2 and Omega-split are similar and lower than that of WBA, although higher than that of uncorrelated arrivals.

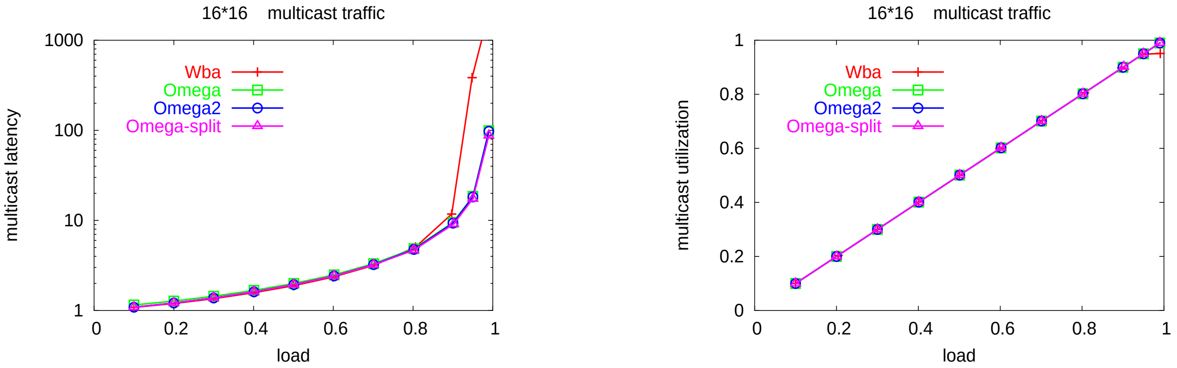


Figure 5.3: Performance of 16×16 switches for uncorrelated multicast traffic with variable fanout.

Figure 5.3 and 5.4 show the performance of a 16×16 switch for uncorrelatd and correlated multiacst traffic, respectively. We see that the performance is similar to that of an 8×8 switch.

Figure 5.5 shows the performance of a 64×64 switch for correlated multicast traffic. Comparing Figures 5.2, 5.4 and 5.5 we see that while the utilization of Omega, Omega2 and Omega-split are almost identical, WBA shows a slight increase in utilization as the switch size is increased from 8 ports to a 64 ports. This is due to the decrease in the effect of correlated traffic with increasing port size of the switch. Since the correlation train size is 16 in all our simulations, on an average the number of new

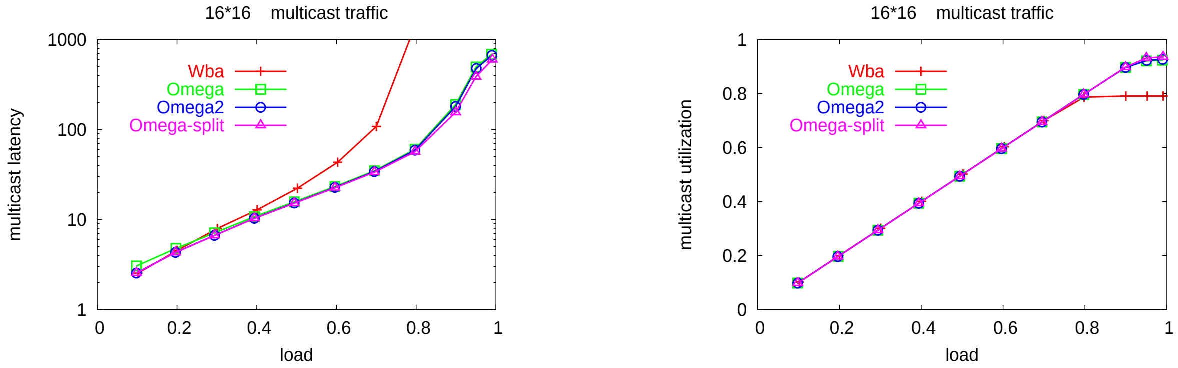


Figure 5.4: Performance of 16×16 switches for correlated multicast traffic with variable fanout.

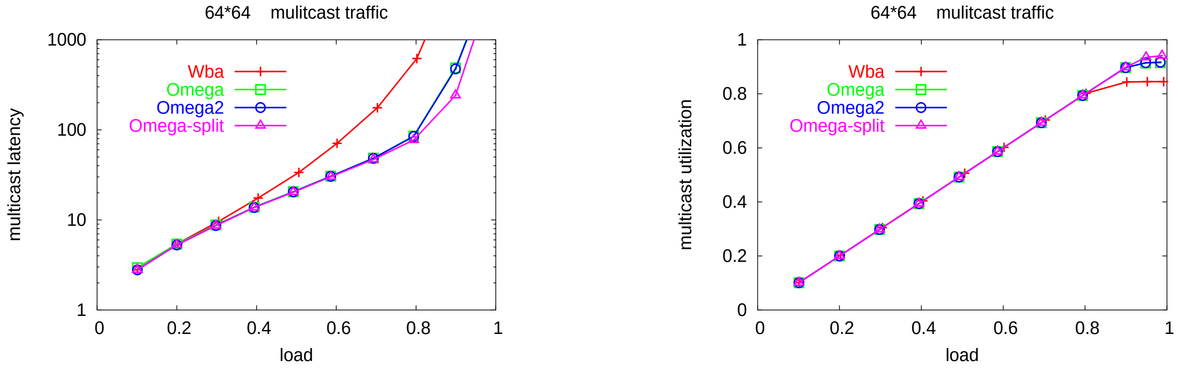


Figure 5.5: Performance of 64×64 switches for correlated multicast traffic with variable fanout.

cells (with possibly different destination list) at the head of the input queue are more in a 64 port switch than in a 16 port switch by a factor of $\frac{64}{16} = 4$. So the output conflicts tend to be randomized better for larger switches. The buffering at output queues in Omega switch reduces the impact of the correlation and thus does not benefit from better randomization of cell destinations.

Constant fanout

In these simulations, the number of destinations (outputs of a switch) that an incoming multicast cell has is fixed. Since the Omega design with round-robin scheduling sends one multicast cell through each copy of its data network, the maximum utilization achieved by it is given by the following formula.

$$\rho = \frac{k \times f}{N} \tag{5.1}$$

where ρ is utilization, k is number of data network copies used in Ω switch and f is multicast cell fanout.

For an 8 port switch of fanout 2, the maximum utilization achieved by Omega is $\frac{2\times2}{8} = 0.5$. Similarly, for 16 port switch with fanout 2 utilization is 0.25 or 25%and for fanout 4 it is 0.50 or 50% for both correlated and uncorrelated traffic.

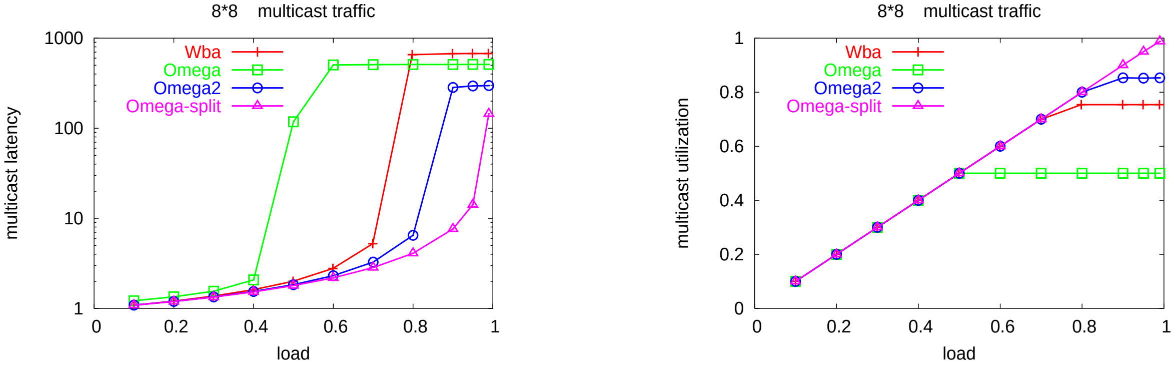


Figure 5.6: Performance of 8×8 switches for uncorrelated multicast traffic with fixed fanout of 2.

Fanout of 2: Figure 5.6 shows the delay and utilization of various designs for uncorrelated arrivals for an 8 port switch. Here, Omega saturates at 50%. Omega2 saturates at around 86% and Omega-split nearly achieves 100%. Splitting the cells makes efficient use of the data paths in the switch fabric. When splitting is not used, the data paths used by some of the cells that could reach the output ports are wasted as all the destinations of these cells could not be satisfied due to output conflicts and hence the cells that could reach their destinations also are not sent. WBA saturates at about 76%. Omega2 and Omega-split have lower delay compared to WBA, specifically, when the loads are in the range 60% - 90%.

Figure 5.7 presents the results for correlated traffic. WBA saturates at about 64% while Omega2 can attain upto 80% and Omega-split saturates at 86%. We notice that correlated arrivals bring down the

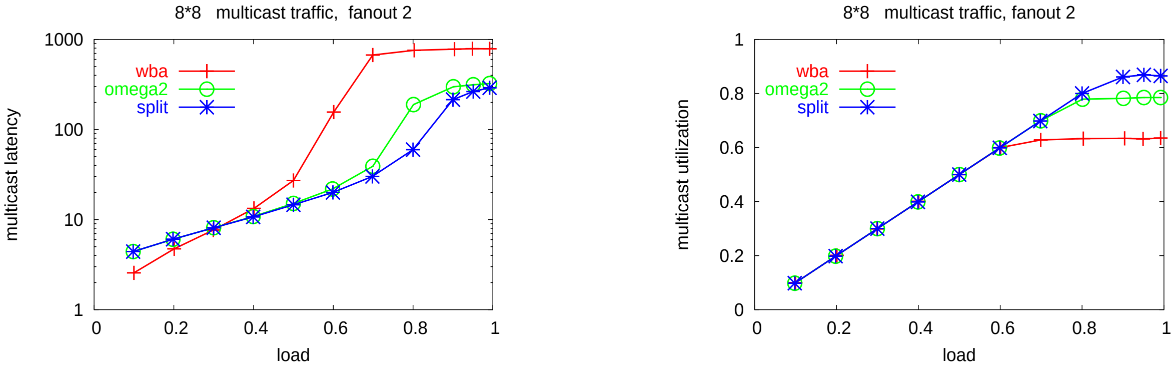


Figure 5.7: Performance of 8×8 switches for correlated multicast traffic with fixed fanout of 2.

switch utilization of all the four designs. The delay curves for Omega2 and Omega-split are significantly lower than that of WBA when the loads are in the range 50% to 90%.

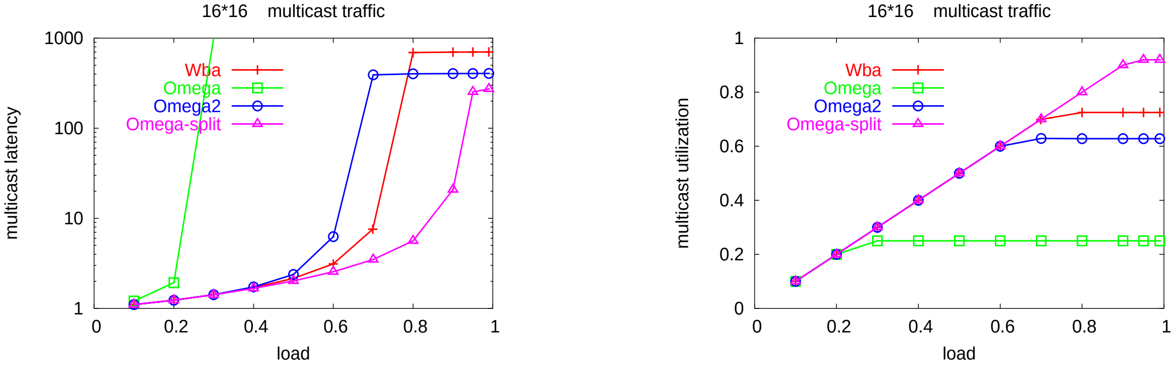


Figure 5.8: Performance of 16×16 switches for uncorrelated multicast traffic with fixed fanout of 2.

Figure 5.8 and 5.9 show the performance of a 16×16 switch for uncorrelatd and correlated multiacst traffic, respectively. Omega-split saturates at about 93%, WBA at about 72%, Omega2 at about 62% for uncorrelated arrivals. For correlated traffic, Omega-split saturates at about 80%, WBA at about 62% and Omega2 at about 61%.

Figure 5.10 shows the performance of a 64×64 switch for correlated multiacst traffic. Here, Omega-split saturates at about 64%, Omega2 at 35% and Omega at 6.25%. On the other hand WBA's utilization is at 60% The utilization achieved by Omega based scheduling policies decrease from an 8 port switch

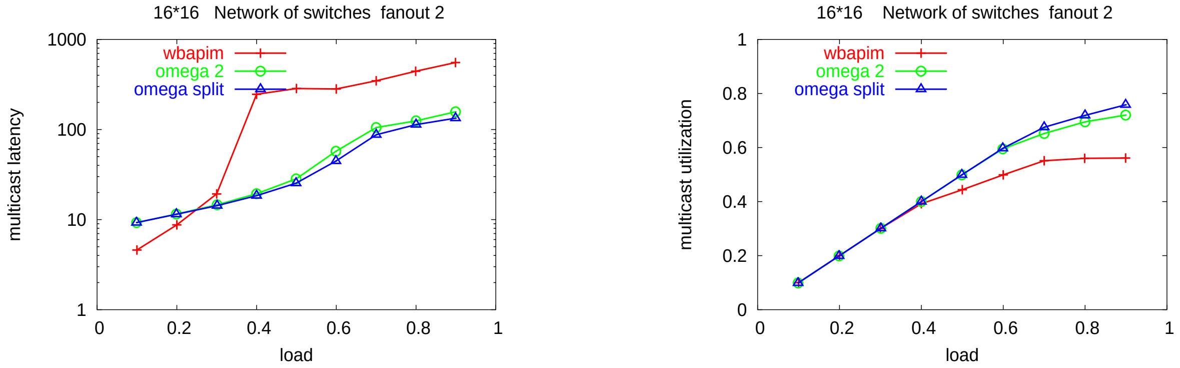


Figure 5.9: Performance of 16×16 switches for correlated multicast traffic with fixed fanout of 2.

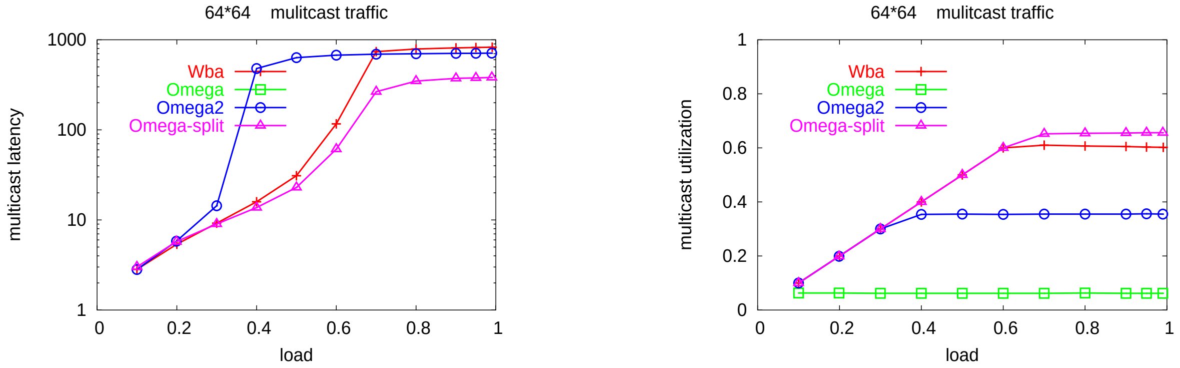


Figure 5.10: Performance of 64×64 switches for correlated multicast traffic with fixed fanout of 2.

to 64 port switch more than that of WBA for multicast traffic with fixed fanout. With increase in number of ports, the cells are routed through an extra stage of switching elements in Omega networks and the probability of path conflicts increases thereby decreasing the switch utilization.

Fanout of 4: Figure 5.11 and 5.12 show the performance of a 16×16 switch for uncorrelatd and correlated multiacst traffic with fixed fanout of 4, respectively. We see from figure 5.11 that Omega-split achieves nearly 100% utilization, WBA about 82%, Omega2 at about 55% for uncorrelated arrivals. For correlated traffic, Omega-split saturates at about 87%, WBA at about 64% and Omega2 at about 55%. Omega saturates at 50% for both kinds of traffic as predicted by equation 5.1.

Figure 5.13 shows the performance of a 64×64 switch for correlated multiacst traffic with fixed

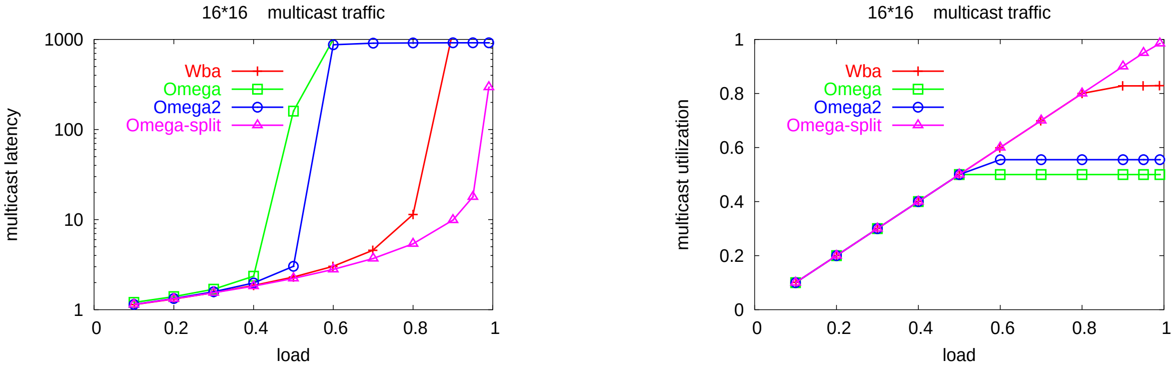


Figure 5.11: Performance of 16×16 switches for uncorrelated multicast traffic with fixed fanout of 4.

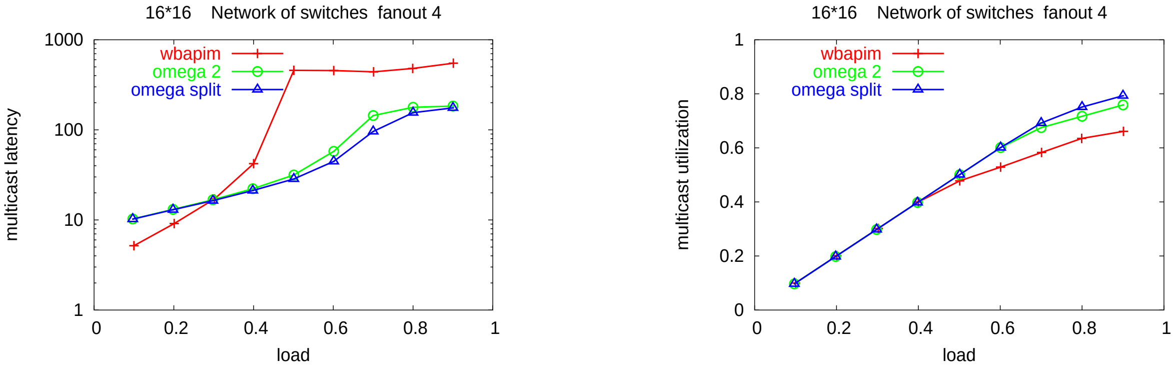


Figure 5.12: Performance of 16×16 switches for correlated multicast traffic with fixed fanout of 4.

fanout of 4. Here, Omega-split saturates at about 74%, WBA at about 63%, Omega2 about 19% and Omega achieves 12.5% utilization.

Fanout of 8: Figure 5.14 shows the performance of a 64×64 switch for correlated multiacst traffic with fixed fanout of 8. Here, Omega-split saturates at about 82%, WBA at about 65% and both Omega and Omega2 achieve 25% utilization.

Comparing Figures 5.5, 5.10, 5.13 and 5.14 we see that with increasing fanout, the switch utilization increases. WBA achieves about 60% for fanout 2, 63% for fanout 4, 65% for fanout 8 and about 83% for variable fanout (with $\frac{N+1}{2}$ destinations on an average). Omega-split achieves about 64% for fanout 2, 74% for fanout 4, 82% for fanout 8 and about 93% for variable fanout. However, Omega2 gives an

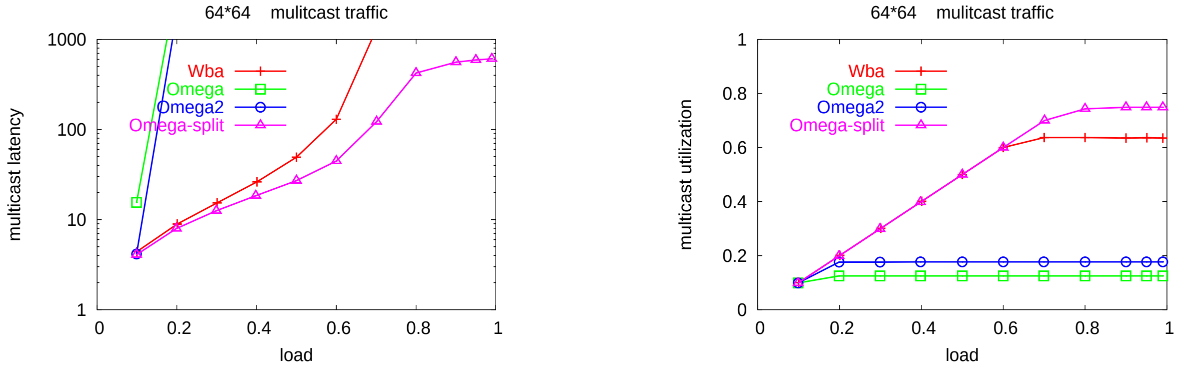


Figure 5.13: Performance of 64×64 switches for correlated multicast traffic with fixed fanout of 4.

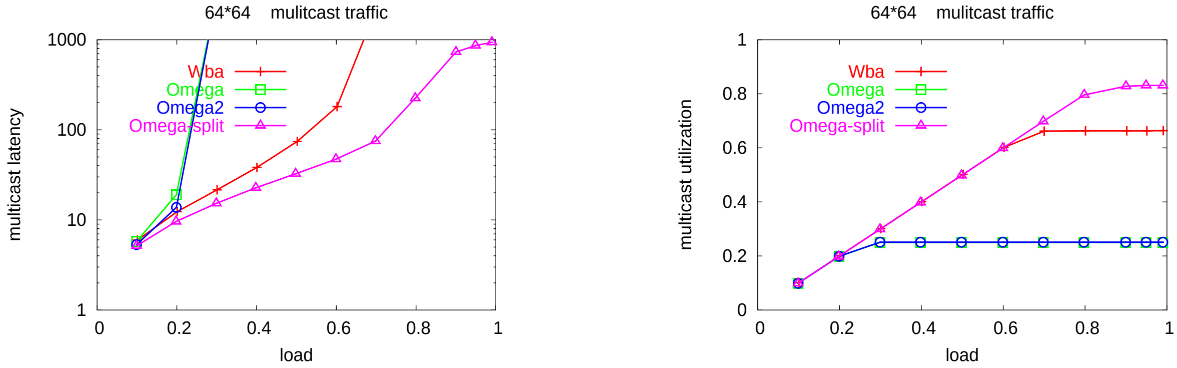


Figure 5.14: Performance of 64×64 switches for correlated multicast traffic with fixed fanout of 8.

utilization of about 35% for fanout 2, but achieves only about 19% for fanout 4 and 25% for fanout 8. Omega2, which acheived higher utilization when the fanout was 2, suffers because with increase in the fanout, the probability of conflicts among the multicast cells with common destinations increases and since a multicast cell is selected only if it could be routed completely to all its destinations in the same slot, hardly more than 2 multicast cells selected by round-robin scheme can be routed in a single slot time. In fact, the extra round of scheduling is not helping Omega2 for multicasts with higher fanouts. So fanout splitting is crucial for traffic with high fanout.

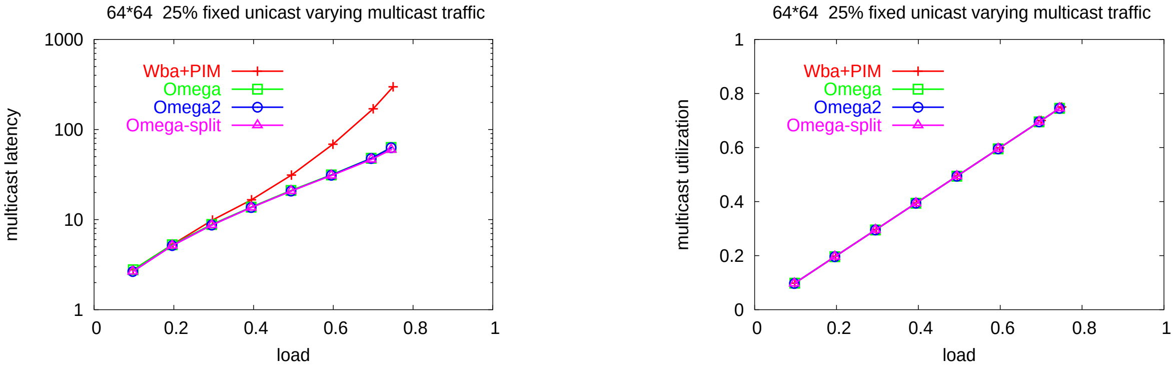


Figure 5.15: Multicast latency, utilization with correlated arrivals of variable fanout, 25% Unicast load.

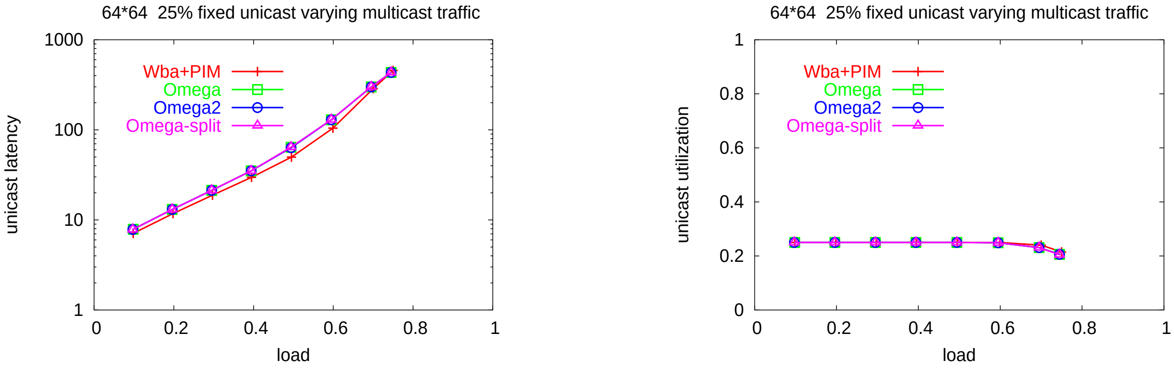


Figure 5.16: Unicast latency, utilization with correlated arrivals of variable fanout, 25% Unicast load.

5.1.2 Mixed Traffic

In this section we present the simulations of 64×64 switch with 25% unicast traffic and varying the multicast traffic. This type of traffic determines the ability of the switch to handle unicast and multicast traffic simultaneously. We simulated 8×8, 16×16 switches also, and the results are similar to those obtained for 64×64 switches.

Figures 5.15, 5.16, 5.17, 5.18, 5.19 and 5.20 show the performance of various designs for correlated multicast traffic with variable fanout and fanout of 2 and 4 for a 64×64 switch in the presence of 25% unicast traffic. We see that all the scheduling policies make use of the remaining bandwidth left after scheduling multicast traffic for the unicast traffic.

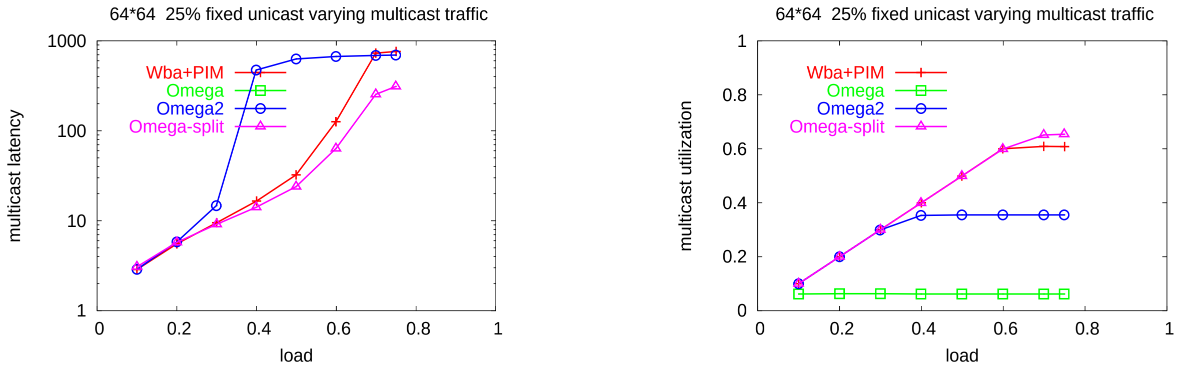


Figure 5.17: Multicast latency, utilization with correlated arrivals of fixed fanout 2, 25% Unicast load.

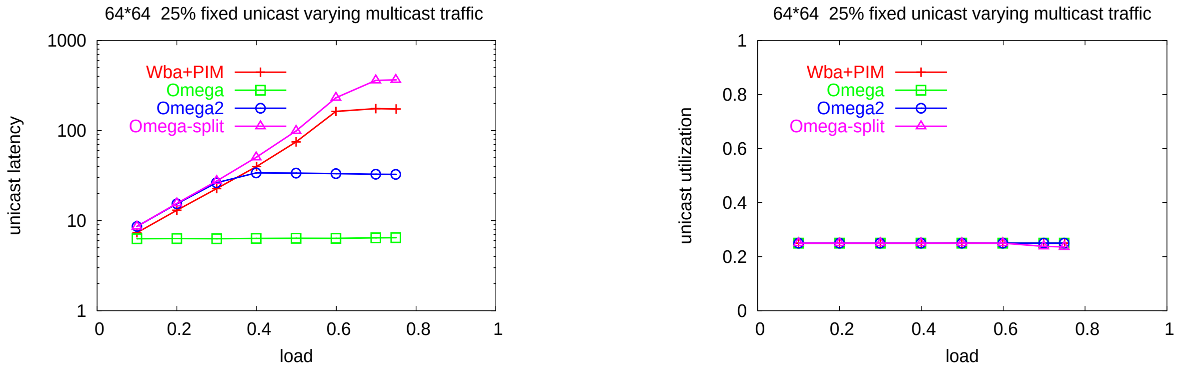


Figure 5.18: Unicast latency, utilization with correlated arrivals of fixed fanout 2, 25% Unicast load.

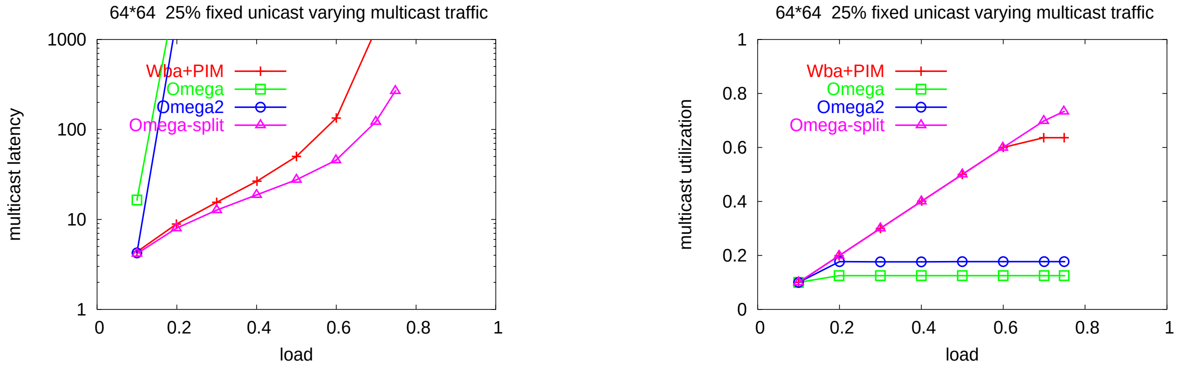


Figure 5.19: Multicast latency, utilization with correlated arrivals of fixed fanout 4, 25% Unicast load.

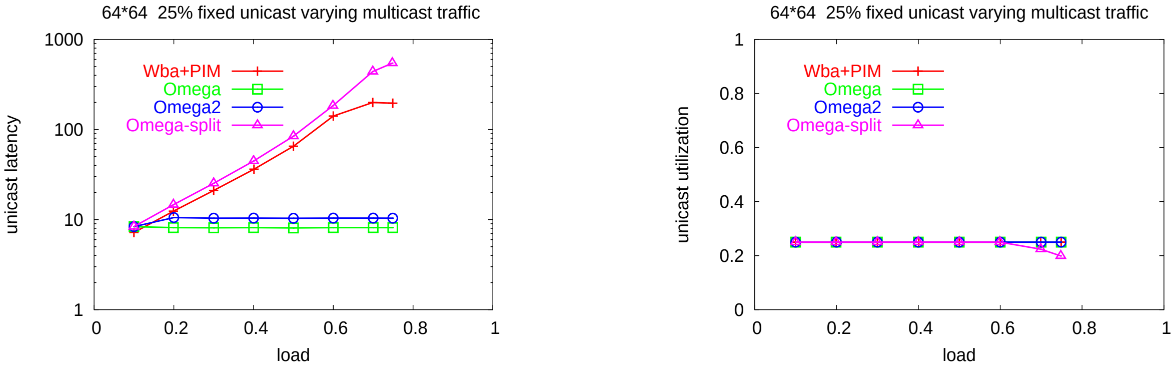


Figure 5.20: Unicast latency, utilization with correlated arrivals of fixed fanout 4, 25% Unicast load.

5.2 Multicycle Scheduling

At very high speeds, it may not be possible to complete the scheduling of the cells for an Ω-switch in a single slot time as the scheduling involves multiple rounds. Instead of increasing the slot time to accommodate the scheduler, we can pipeline the scheduling of the cells, thereby keeping the scheduler in sync with the line rates and switch fabric capacity. Hence, the scheduler takes multiple slot times to schedule the cells. This enables us to deliver the cells at line rates though it takes multiple cycles in making a decision to schedule them.We have considered the following scenario to evaluate the effect of pipelining on the latency and throughput of the switch.

A simple way to implement a pipelined effect is to assume that there are as many stages in the pipeline as the number of copies of Ω network in the data path of the switch fabric. During each slot time the inputs send their requests to the scheduler. After n slot times, the inputs would know the outcome of their requests and those successful would send those cells which match the granted requests through the data network of the switch fabric in that time slot, which itself can be considered as another stage in the pipeline. Hence, at any time t the scheduler would be processing requests from n previous time slots, that is requests from $t - n$ to $t - 1$ slots, where n is the length of the pipeline.

By this, subsequent requests by the inputs are pipelined and once the succesfull requests are known the input ports choose those cells that match the granted requests and route them through the data network.

In order to be conservative in simulating the effect of pipelining the scheduler on the throughput and latency, we cancel all the subsequent requests of an input port whose requests are not granted due to blocking nature of Ω network at any of the stages in the pipeline. That is, if an input port sends its request to the scheduler at time t and the request is denied due to conflicts in the pipeline, all requests made after time t are cancelled and considered again after time $t + n$. This would delay the internal scheduling conflicts to later slots. Though this might increase the latency of the cells, we do not expect much change in the utilization of the switch.

We have simulated the pipeline effect with correlated traffic of fixed fanout 2 for an 8 port switch for camparing it with the original 8×8 Ω switch with multicast traffic. The delay seen by a cell in an otherwise empty switch increases from 2 slots (1 for scheduling and 1 for routing) in earlier simulations to n+1 slots. For WBA the cell delay is still 2 slots in the absence of contention or waiting. Figure 5.21 shows the effect of pipelining on Omega2, Omega-split for an 8×8 Ω switch. Referring to fig. 5.7, we see that latency increases at lower loads by 10 - 20% but for higher loads the latencies are similar. However, the increase in the latency at lower loads is not considerable because its less than 10 time

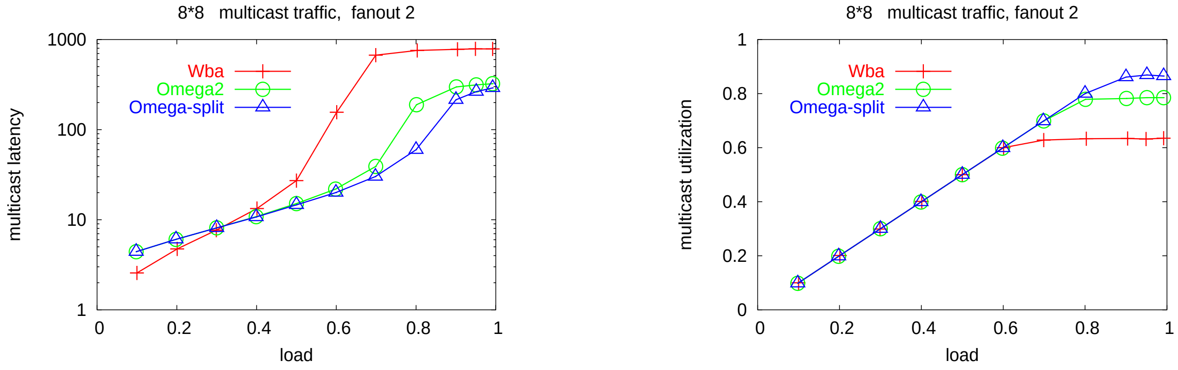


Figure 5.21: Effect of pipelining on 8×8 switches for correlated multicast traffic with fixed fanout of 2.

slots. At higher loads,buffering time of cells overlaps with the extra time spent on scheduling. So the impact of increased scheduling time is muted. The throughput is slightly lower because of the increased cell scheduling time.

5.2.1 Performance Analysis of Large Switches

In this section we present the simulations of multicast traffic for large switches i.e., switches with ports 512 and 1024. We also examine the effect of the number of data networks in the switch fabric.

We have simulated 512×512 and 1024×1024 Ω- and crossbar-based switches with separate buffers for unicast and multicast traffic. While each input queue of crossbar can buffer up to 256 multicast cells and 256 unicast cells, input queues of Ω can buffer 128 unicast cells and 128 multicast cells at the input and 128 cells at the output. All the graphs are for correlated traffic. In these simulations, we vary the number of copies of data network in Ω-switch designs to determine the benefit of more copies of data network. Since Omega-split is the only Omega design that appeared competitive as the ratio $\frac{fanout}{switchsize}$ decreased, we simulated only Omega-split design for 512 and 1024 port switches.

The simulation for 512 and 1024 port switches were run for 3000 cycles (averaged over 10 runs) with a warmup of 1000 cycles.We use the following notations for all the plots in this section.

Split k : The scheduling policy using fanout splitting with k copies of Omega network. The minimum cell latency is $k+1$ slot times.

WBA: The WBA multicast scheduling policy over crosbars[11].

Observations

We observe that Split-3 and Split-4 behave similarly for both 512 and 1024 port switches, that is, there is very little increase in utilization with 4 copies over 3 copies. However, the utilization is improved by 20 percentage points when 3 copies (split-3) are used instead of 2 (split-2). The differences in the performance of 3 and 4 copies decrease with increasing fanout. For fanout 8 and above they are almost

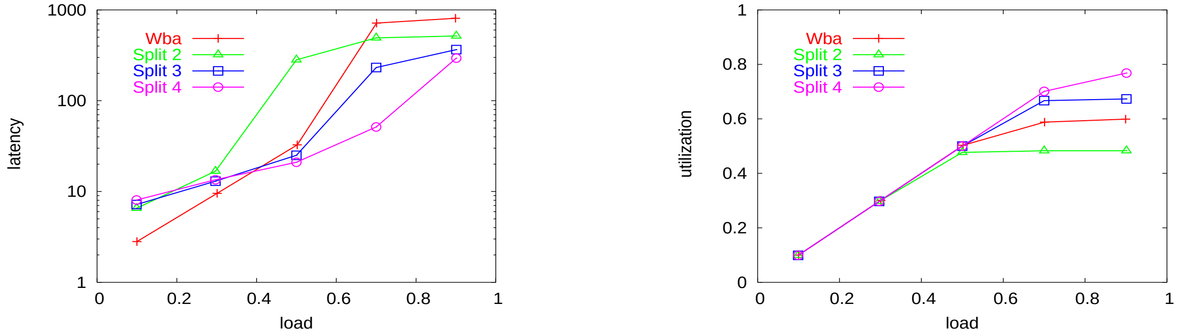


Figure 5.22: Performance of 512×512 switches for correlated multicast traffic with fixed fanout of 2.

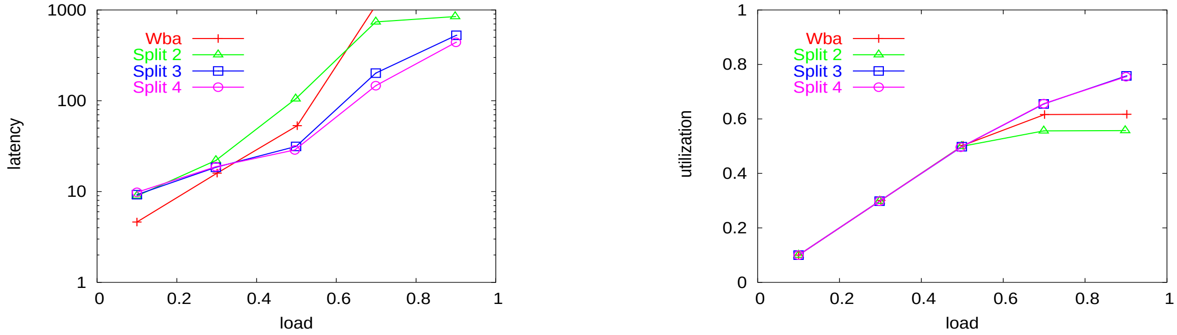


Figure 5.23: Performance of 512×512 switches for correlated multicast traffic with fixed fanout of 4.

identical. Hence the difference is prominent only in fanout 2 and 4.

High throughput for 3 copies is achieved because there were many input ports that did not deliver their multicast cells to any of its destinations through the 2 copies because of internal path conflicts. There is no significant improvement with 4 copies over 3 copies because of the following reasons.

1. All or most of the input ports were able to send their multicast cell to atleast one of their destinations with 3 copies.

2. The remaining input ports (i.e., those which didn't send its multicast cell to any of their destinations through the 3 copies) have output conflicts with the cells already sent. That is, they may be requesting same output ports as the cells that have already reached the output ports using the 3 copies. And hence, they would be rejected once the output buffers saturate or would be buffered

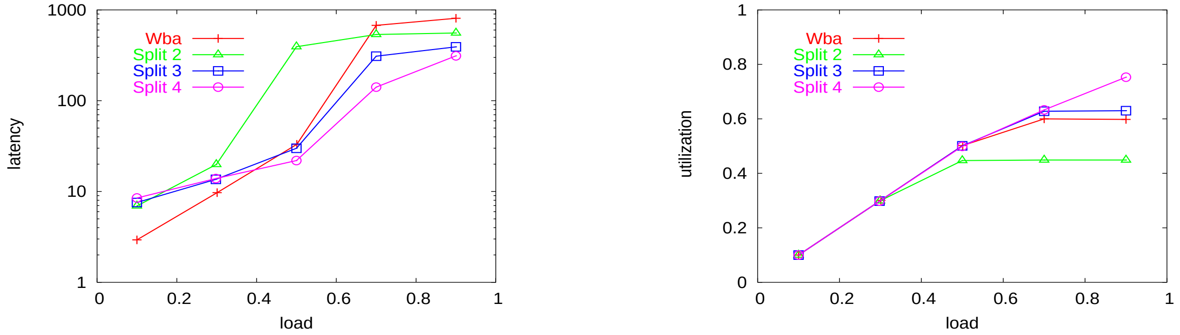


Figure 5.24: Performance of 1024×1024 switches for correlated multicast traffic with fixed fanout of 2.

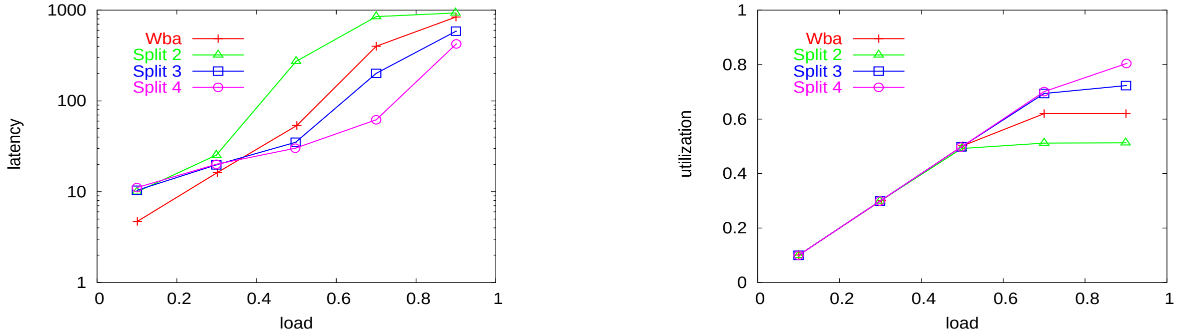


Figure 5.25: Performance of 1024×1024 switches for correlated multicast traffic with fixed fanout of 4.

at the output and hence do not increase the switch utilization.

Performance could be improved for 4 copies if we consider those input ports that have split their cells during scheduling and have succeeded in sending some of their destinations through a copy of the network for routing through the remaining copies of the data network also.

We also see that the Split-3 and Split-4 have lower latencies than Split-2. This is directly related to high throuput for 3 and 4 copies over 2 copies. Because of high switch utilization, cells get delivered faster and hence latencies are low.

We notice that the fanout splitting policies over Ω switch require 3 data networks to outperform WBA for larger switches of size 512 and above. In terms of number of crosspoints Split-3 requires fewer than 11% ($\frac{3\times\frac{512}{2}\times 9\times 4}{512\times 512}$) of the crosspoints required by a crossbar.

Chapter 6

Switched Networks

Most published studies evaluate proposed switch designs as single components, but do not evaluate their part in a networked environment. We used Omega and WBA switches as building blocks to form a switched network by interconnecting multiple switches. We have evaluated the performance of the complete switched network built with each type of switch. In particular, we analyzed the performance of the switches for 2 network topologies shown in Figures 6.1 and 6.2. Figure 6.2 presents a general topology for SANs. On the other hand the topology in figure 6.1 provides the functionality of a 16×16 switch. We have chosen Figure 6.1 for comparison with the single switch results presented earlier.

Simulation

We have modified the simulator to simulate the interconnected switches. The program takes the number of switches, input ports, output ports, the size of the switch, the traffic loads as a fraction in the range [0,1] and the type of scheduling policy and the type of switch design to be used as parameters. Each switch also reads in a file as an input which describes its interconnection and hence defines the topology of the network. We modified the Omega.java and Wba.java described in Chapter 4 to exist as independent objects. We developed another main class called Network.java which simulates the network by initializing all the switches and passing control to each of them appropriately.

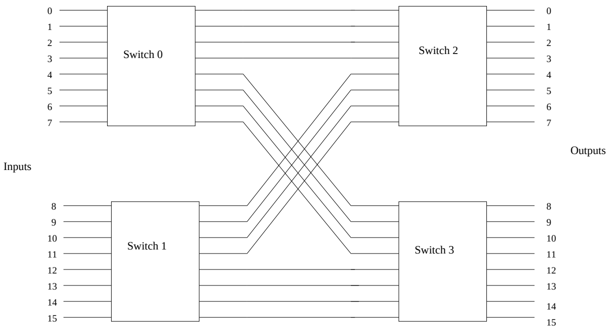


Figure 6.1: Topology of the simulated network 1.

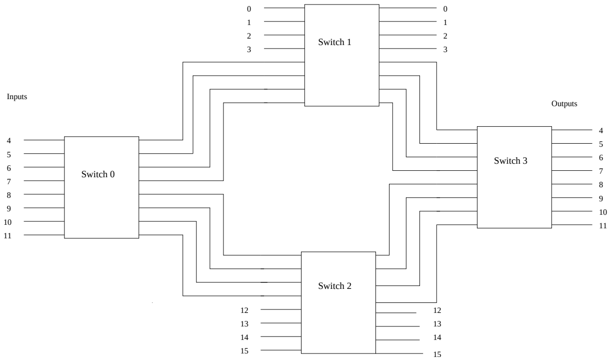
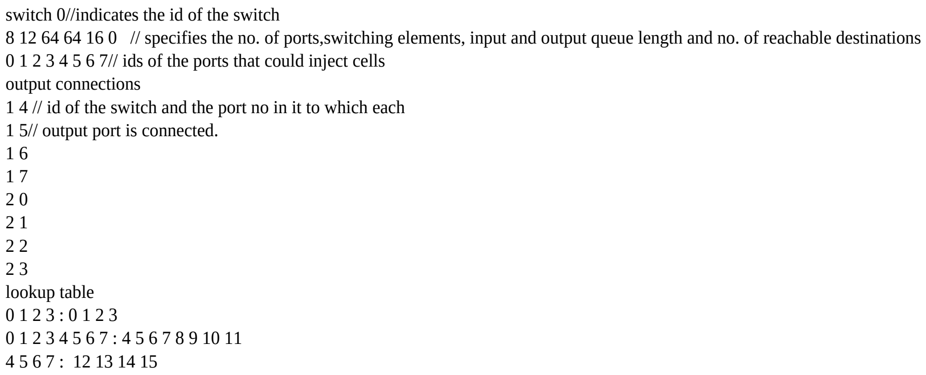


Figure 6.2: Topology of the simulated network 2.

The topology of the network is described by several input files (one per switch), which specify how each switch is connected to its neighbors. Each input file specifies the inputs of the switch that can inject cells into the network which are the network inputs. The file also specifies the neighboring switch and port each of its output port is connected to. A sample input file for Switch 0 in network Figure 6.2 is shown in Figure 6.3.

In the above example, the first line specifies the number of ports, switching elements, input and output buffer size, number of network outputs reachable through switch 0 and the numerically least id of the reachable network outputs. The second line specifies the switch inputs that are also the network inputs. Although the above format represents reachable destinations by each switch to be contiguous, it can be modified to remove the restriction. The switches themselves are simulated to handle any com-



```
switch 0//indicates the id of the switch
8 12 64 64 16 0   // specifies the no. of ports,switching elements, input and output queue length and no. of reachable destinations
0 1 2 3 4 5 6 7// ids of the ports that could inject cells
output connections
1 4 // id of the switch and the port no in it to which each
1 5// output port is connected.
1 6
1 7
2 0
2 1
2 2
2 3
lookup table
0 1 2 3 : 0 1 2 3
0 1 2 3 4 5 6 7 : 4 5 6 7 8 9 10 11
4 5 6 7 :  12 13 14 15
```

Figure 6.3: Input file for Switch 0 in network 6.2.

bination of outputs. Consecutive lines in the input file under the heading output connections specifies the neighboring switch and port number each of Switch 0's output port is connected to. For the above example, output port 0 of switch 0 is connected to input port 4 of switch 1. A $-1-1$ in this line means that the output port of the switch is also a network output. That is, for Switch 2 and Switch 3 all lines under output connections will have $-1-1$ as they are the network outputs.

The lookup table describes all the network outputs that can be reached through the local outputs of the switch. For example, input file in Figure 6.3 specifies that Switch 0 can reach network outputs 0, 1, 2 and 3 through its local outputs 0, 1, 2 or 3. Similarly network outputs 4, 5, 6, 7, 8, 9, 10 and 11 can be reached by going to either 0, 1, 2, 3, 4, 5, 6 or 7 of that particular switch.

We have grouped the network outputs based on the switches they are connected to. Each cell has the list of all the destinations it want to go to in its header. When a cell comes into a switch input port, we determine the different groups its destination vector is comprised of and then for each group it chooses one local output port of that particular switch randomly from the table. For example, if we had a cell with destinations 2, 7 and 9 we see that network output 2 belongs to group 1 and 7, 9 belong to group 2. Now the cell randomly picks the outputs it would use to go to these groups 1 and 2. One possibility is it picks switch output 1 for group 1 and switch output 2 for group 2. another possibility is it picks switch output 1 for both groups. (There are numerous such possibilities.) This routing scheme doesn't minimize the number of switch outputs used.

6.1 Performance Evaluation

We present the simulations of two networks for multicast traffic of fanout 2 and 4 in this section. Omega designs use two copies of data network and the scheduling time is 2 cycles, while that of WBA is 1 cycle.

6.1.1 Network 1

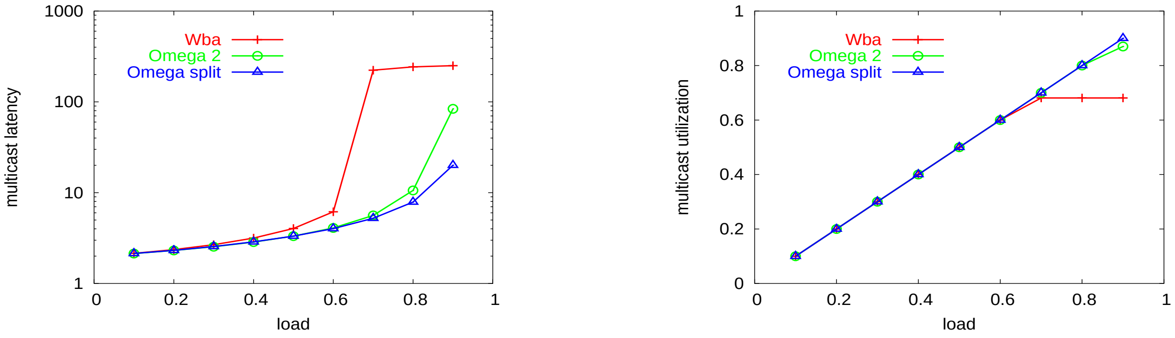


Figure 6.4: Performance of the switched network in Figure 6.1 for uncorrelated multicast traffic with fixed fanout of 2

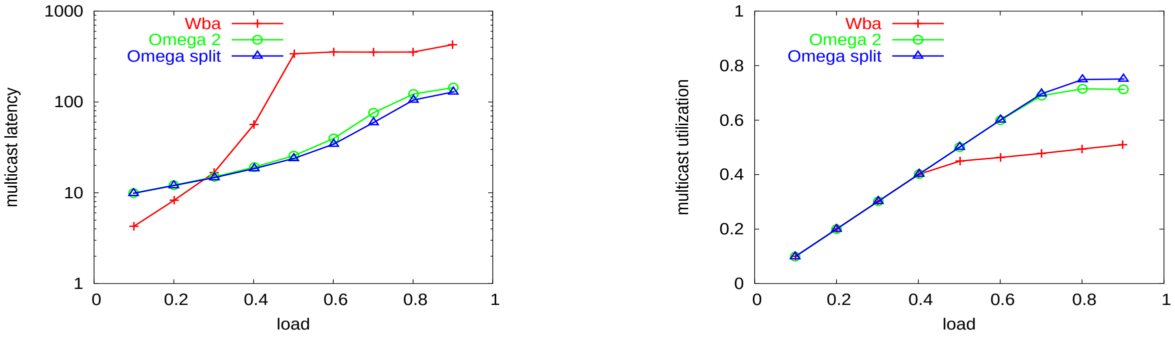


Figure 6.5: Performance of the switched network in Figure 6.1 for correlated multicast traffic with fixed fanout of 2

Figures 6.4 and 6.5 present the network utilizations and latencies achieved by the switches for network in 6.1 for multicast traffic of fanout 2. From Figures 6.4 and 6.5 we see that Omega2 performs similar to Omega-split for both correlated and uncorrelated traffic in terms of throughput and latency. Comparing the results with a single 16×16 port switch from Figures 5.8 and 5.9 we see that Omega2 achieves a utilization of over 80% for uncorrelated multicast traffic of fanout 2, before starting to lose cells whereas a single 16×16 switch gives about only 60% utilization with Omega2. For correlated traffic it achieves about 70% utilization whereas in a single switch it gives about 60% utilization. The latency curve for Omega2 almost traces the latency curve for the Omega-split design. Also, Omega-split design performs similarly in terms of throughput and latency for a single 16×16 switch as well as for interconnected switches as in Figure 6.1.

The network utilization depends on utilization achieved by each of the switches in the network. By arranging the switches in stages we have reduced the effective fanout of the cells passing through each stage of the switches. With fanout 2, a cell at Switch 0 may need to use only one output of Switch 0 (because both of its destinations are connected to the same switch in the next column) or two outputs of Switch 0 (because one destination is reachable through Switch 3 and the other through Switch 4). The probability of former is $\frac{7}{15}$ and that of the later is $\frac{8}{15}$. So the effective fanout of cells seen by Switch 0 and Switch 1 is $1 \times \frac{7}{15} + 2 \times \frac{8}{15} = \frac{23}{15} = 1.53$. And the effective fanout of cells seen by switches 2 and 3 is $\frac{2}{1.53} = 1.3$. This attributes to the increase in performance of Omega2 as there are cells of fanout 1 and 2, whereas in single large switch, all cells have fanout of 2. Omega-split is not effected because it already employs fanout splitting and in a steady state there would be cells with multiple fanouts. By the above argument, the network utilization for multicast traffic of fanout 2 is dependent on the performance of an 8×8 switch with multicast traffic of fanout $\frac{23}{15}$. We see that the network utilization achieved for the network in Figure 6.1 is in fact comparable to the performance of an 8×8 switch with multicast traffic of fanout 2 in Figure 5.7.

WBA based network in Figure 6.1achieves about 70% utilization for uncorrelated traffic which is the same as that of a single 16×16 switch. However, it achieves only 40% switch utilization for correlated traffic while that of a single 16×16 WBA switch is about 60%. Since the effective fanout of cells at switches 0 and 1 is 1.53 we know that each multicast cell at the input of switches 0 or 1 becomes 1.53 cells at the inputs of switches 2 and 3. The input buffers at switches 2 and 3 are not able to accommodate the number of multicast cells delivered by switches 0 and 1 due to this expansion. The simulations show that large number of cells are lost as they could not enter the input buffers of switches 2 and 3, although they were delivered by switches 0 and 1. For example, in the simulation of the network with a network load of 0.6, fanout 2 and correlation train of size 16, we saw that for 100,000 slots around 480,000 multicast cells were injected into switches 0 and 1 and 735,000 are delivered by switches 0 and 1 to switches 2 and 3. However, out of these 735,000 cells only 568,000 cells enter the input buffers of switches 2 and 3 and the remaining 167,000 cells are lost. Switches 2 and 3 deliver all the cells in their buffers, which is around 738,000 cells that account for only 46% utilization.

For a mulitcast load of 0.4 with fanout 2 on network in Figure 6.1, the cell rate (rate at which cells are injected in to the inputs) at switches 0 and 1 is $\frac{0.4}{2} = 0.2$. Since the effective fanout at these switches is 1.53, the cell rate at switches 2 and 3 is $2 \times 1.53 = 0.31$. This is the cell rate that a single 8×8 switch having a load of 0.64 multicast traffic with fanout 2 also starts to lose cells. This shows the inherent limitaion on the cell rate that the WBA can sustain on the crossbars irrespective of the fanout for correlated traffic.

When the cell rate at the inputs is high, the fanout of the cells is very low as cell rate and fanout are inversely related. The low fanout brings up the head-of-line problem for crossbar designs. The theoretical limits for head-of-line problem is 58% for uncorrelated traffic. Correlated traffic exacerbates the head-of-line problem and further reduces the achievable utilization. Hence, we see that the utilization achieved by WBA is only about 40% for network in Figure 6.1 for correlated trffic of fanout 2 while that of a single 8×8 switch is around 64%.

This clearly demonstrates the inherent weakness of input buffered crossbar designs for correlated traffic. This also illustrates the advantage of using limited output buffering, especially for correlated multicast traffic. For this reason, the Ω switch designs take advantage of reduced effective fanout in the traffic that occurs as cells go through multiple switches.

The latency curve for uncorrelated traffic is similar to that of a single 16×16 switch but the latencies in Figure 6.5 are higher than that of a single switch in Figure 5.9.

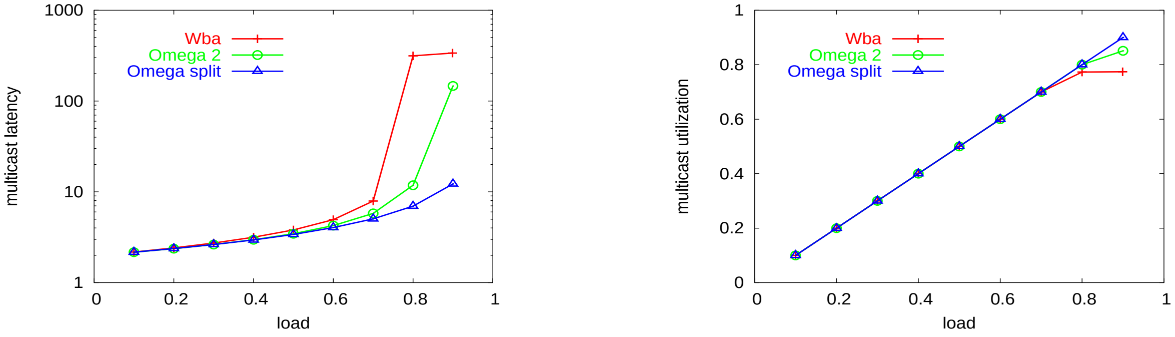


Figure 6.6: Performance of switch based network for topology in Figure 6.1 for uncorrelated multicast traffic with fixed fanout of 4

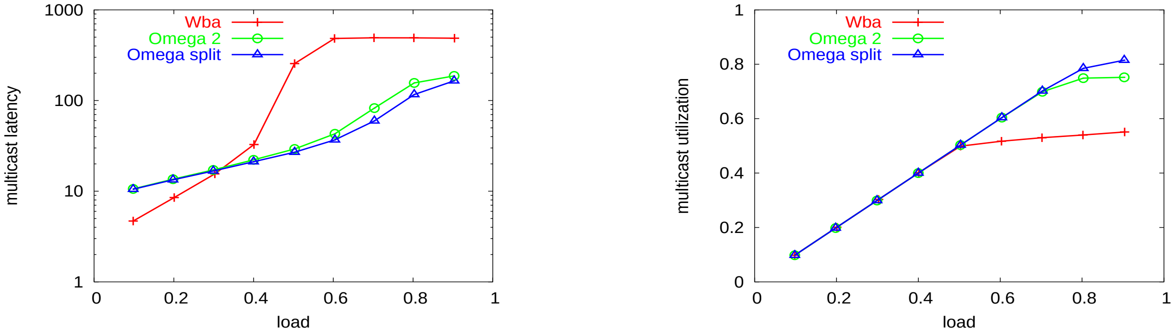


Figure 6.7: Performance of switch based network for topology in Figure 6.1 for correlated multicast traffic with fixed fanout of 4

Figures 6.6 and 6.7 present the network utilizations and latencies achieved by the switches for network in 6.1 for multicast traffic of fanout 4. From Figures 6.6 and 6.7 we see that Omega2 performs similar to that of Omega-split for both correlated and uncorrelated traffic in terms of throughput and latency. Comparing the results with a single 16×16 port switch from Figures 5.11 and 5.12 we see that Omega2 achieves a utilization of over 80% for uncorrelated multicast traffic of fanout 4, before starting to lose cells whereas a single 16×16 switch gives about only 58% utilization with Omega2. For correlated traffic it achieves about 75% utilization whereas in a single switch achieves about 58% utilization. The latency curve for Omega2 almost traces the latency curve for the Omega-split switch based network. Also, Omega-split design performs similarly in terms of throughput and latency for a single 16×16 switch as well as for interconnected switches as in Figure 6.1. WBA achieves about 78% utilization for uncorrelated traffic which is the same as that of a single 16×16 switch. For correlated traffic , however, WBA achieves only 50% switch utilization while it achieves 64%utilization if a single 16×16 switch is used. The reasons for underperformance are the same as those explained for fanouts of 2. The latency curve for uncorrelated traffic is similar to that of a single 16×16 switch but the latencies in Figure 6.7 are higher than that of a single switch in Figure 5.12.

6.1.2 Network 2

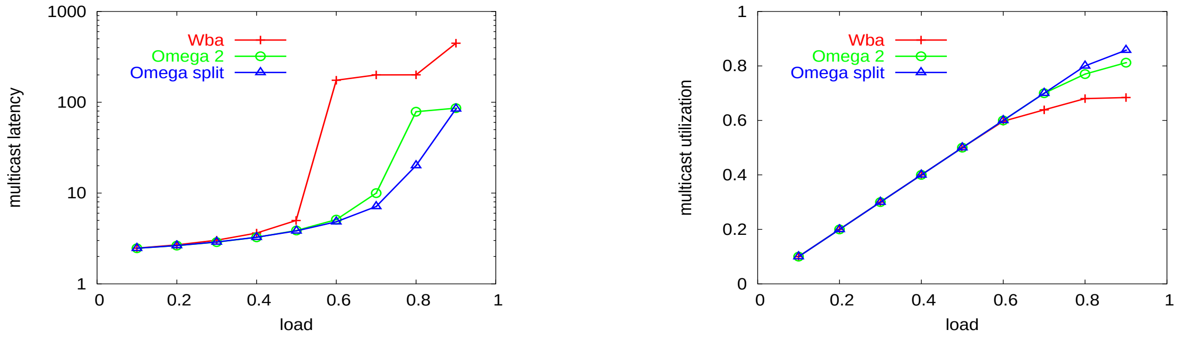


Figure 6.8: Performance of the switched network in Figure 6.2 for uncorrelated multicast traffic with fixed fanout of 2

Figures 6.8 and 6.9 present the network utilizations and latencies achieved by the switches for network in 6.2 for multicast traffic of fanout 2. From Figure 6.8 we see that the Omega-split, Omega2, WBA scheduling policies acheive about 80%, 70%, 60% utilization for uncorrelated multicast traffic of fanout 2 before they start losing cells. For correlated traffic of fanout 2, Omega-split, Omega2, WBA

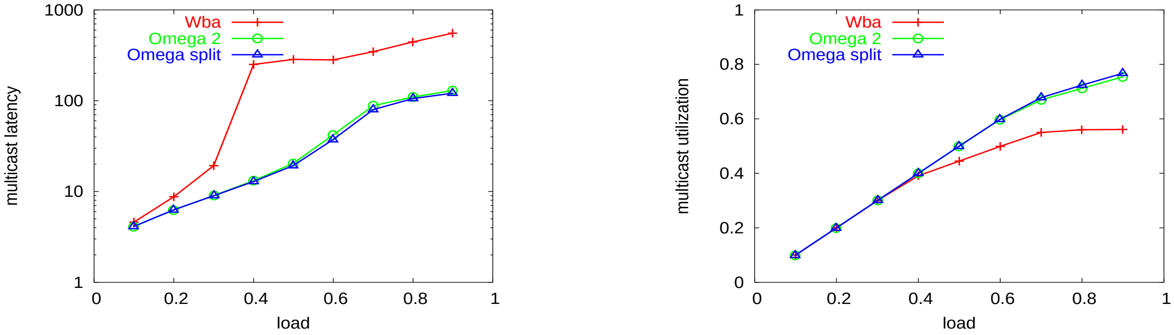


Figure 6.9: Performance of the switched network in Figure 6.2 for correlated multicast traffic with fixed fanout of 2

achieve a utilization of 60%, 60% and 40% before they start losing cells, see Figure 6.9.

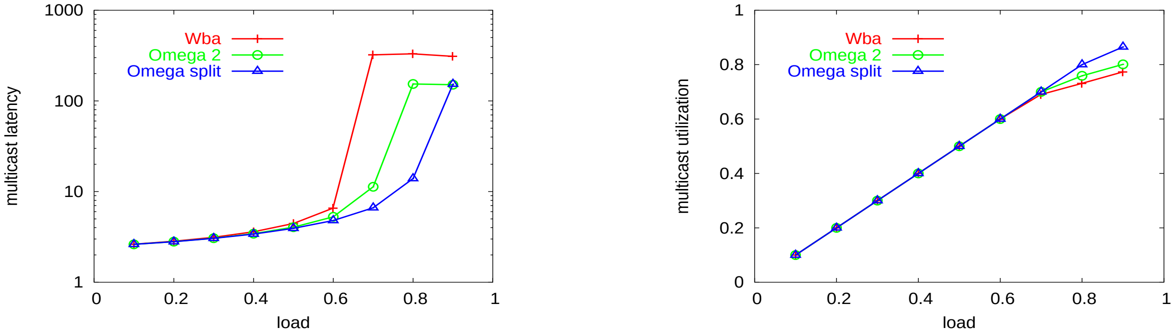
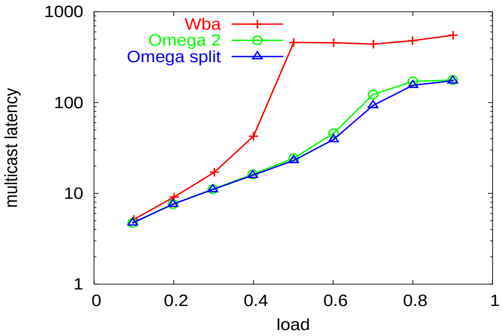


Figure 6.10: Performance of the switched network in Figure 6.2 for uncorrelated multicast traffic with fixed fanout of 4

Figures 6.10 and 6.11 show the network utilizations and latencies achieved by the switches for network in 6.2 for multicast traffic of fanout 4. From Figure 6.10 we see that the Omega-split, Omega2, WBA scheduling policies achieve about 80%, 70%, 70% utilization for uncorrelated multicast traffic of fanout 4 before they start losing cells. For correlated traffic of fanout 4, Omega-split, Omega2, WBA achieve a utilization of 60%, 60% and 40% before they start losing cells, ref Figure 6.11. It can also be seen that Omega2 and Omega-split behave similarly in terms of throughput and latency for correlated traffic.



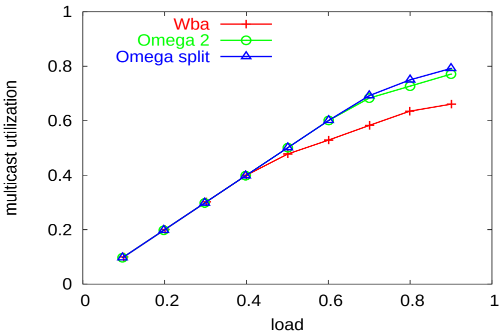


Figure 6.11: Performance of the switched network in Figure 6.2 for correlated multicast traffic with fixed fanout of 4

Chapter 7

Conclusions

The increasing demand for network bandwidth for various multicast applications requires storage area networks based on high-speed multicast switches. These switches should handle both unicast and multicast traffic efficiently. In this thesis, we have presented the design of a multicast switch based on multistage interconnection network with input and output buffers. Being based on multistage network, the design is cost effective and scalable. To overcome the possible contention for paths through the switch fabric, we used multiple copies of the well known Omega network. We have used a limited form of output queueing to handle ouput conflicts. To resolve contention for paths through the switch fabric, we have used a hardware based cell selection strategy. The proposed design simplifies the switch fabric and ouput queueing and makes the cell selection at inputs more complex.

We have illustrated three possible designs based on the cell selection policy. They are Omega which has a simple round-robin scheme, Omega2 which has an additional round of scheduling in addition to the round-robin scheduling, and Omega-split which is basically Omega2 with cell splitting. We have developed a modular simulator in java to evaluate the performance of the switches for the proposed scheduling policies. The program can simulate the above three scheduling policies over Omega network and crossbar based WBA for various switch sizes and traffic types. We have observed that the Omega-split outperforms WBA on crossbar for smaller switch sizes like $8 \times 8, 16 \times 16$ and 64×64. It achieved almost 100% utilization for mixed traffic, that is in the presence of both unicast and multicast traffic.

We have also analyzed the effect of the number of the data networks for Omega based switches with large number of ports. For switches with 512 and 1024 ports, Omega based designs needed 3 data networks to outperform WBA on crossbar. All the scheduling policies performed similarly with multicast traffic of variable fanout. However, the scheduling policies reached their limitations in the presence of low fanout multicast traffic.

We have analyzed the performance of the switches in the context of switched networks for two network topologies. In particular, we simulated a 16×16 switch by interconnecting four 8×8 switches and observed that the throughput achieved is more than that of a single 16×16 switch for Omega based designs. Moreover, Omega2 switches performed as well as the Omega-split switches in terms of utilization and latency. Another network topology we have simulated is a general topology which represents a more realistic interconnection of switches especially for SANs. Another contribution of our study is that WBA which uses crossbars as switch fabrics suffers as its not able to sustain the cell rate for low fanout multicast traffic that occurs as cells go through multiple switches in a network.

Future work

The proposed Ω designs are atttractive in terms of cost, performance and scalability compared to current designs. One of the common deficiencies of current studies is that single switch design is studied and conclusions are drawn. As our simulations of networks indicate, this could lead to misleading conclusions. So it is important to conduct realistic simulation. Future work in this direction will be to simulate larger switched networks with more realistic traffic. For example, a multicast from servers to storage devices followed by a unicast confirmation for each recipient storage unit to the sender of the multicast. We plan to study these aspects in future.

Bibliography

[1] J. turner, “Design of a broadcast packet switching network,” IEEE Trans. Commun., vol. 36, 1988, pp. 734-743.

[2] T. Lee, “Nonblocking copy networks for multicast packet switching,” IEEE J. Select. Areas of Commun., vol. 6, 1988, pp. 1455-1467.

[3] S. Shimamoto, W. Zhong, Y. Onozato and J. Kaniyil, “Recursive copy networks for large multica st ATM switches,” IEICE Trans. Commun., vol. E75-B, 1992, pp. 1208-1219.

[4] K. Schrodi, B. Pfeiffer, J. Delmas and M. DeSomer, “Multicast handling in a self-routing switch architecture,” Proc. of ISS’92, pp. 156-160.

[5] J. Chao and B. Choe, “A large-scale multicast output buffered ATM switch,” Proc. of GLOBECOM’93, pp. 34-41.

[6] R. Cusani and F. Sestini, “A recursive multistage structure for multicast ATM switching,” Proc. of INFOCOM’91, pp. 1289-1295.

[7] J. Turner, “An optimal nonblocking multicast virtual circuit switch,” Proc. of INFOCOM’94, pp. 298-305.

[8] J. S. Turner, “Design of a Broadcast Packet Switching Network,” IEEE Trans. Commun. vol. 36, pp734-743, June 1998.

[9] Y. Xiong and L. Mason “Multicast ATM switches using buffered MIN structure: A performance study,” INFOCOM’97 vol. 3 pp. 924-931.

[10] Rajendra V. Boppana, C. S. Raghavendra, “Designing efficient Benes and Banyan based input buffered ATM switches,” ICC’91.

[11] Balaji Prabhakar, Nick McKeown, Ritesh Ahuja, “Multicast Scheduling for Input-Queued Switches,” IEEE Journal on Selected Areas in Communications, vol 15, No. 15, pp. 885-866, June 1997.

[12] Nick McKeown and Balaji Prabhakar, “Scheduling Mulitcast cells in an Input-Queued Switch,” INFOCOM ’96 vol. 1 pp. 271-278.

[13] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, “High Speed Switch Scheduling for Local Area Networks,” ACM Transactions on Computer Systems 11, 4, November 1993, pp. 319-352.

[14] M. J. Karol, M. G. Hluchyj, S. P. Morgan, “Input Versus Output Queueing on a Space-division Packet switch,” IEEE Transactions on Communications, December 1987, pp. 1347-1356.

[15] Y. Yeh, M. Hluchyj, and A. Acampora, “The Knockout Switch: A Simple, Modular Architecture for High-Performance Packet Switching,” IEEE Journal on Selected Areas in Communications, Vol SAC-5, No. 8, October 1987, pp. 1274-1283.

[16] H. S. Kim and Alberto Leon-Garcia, “A self routing multistage switching network for broadband ISDN,” IEEE J. Select. Areas Commun., vol. 8, No. 3, Apr. 1990.

[17] Tamir. Y., G. Frazier, “High-performance Multi-queue buffers for VLSI Communication Switches,” Proc. of 15th Ann. Symp. on Computer Architecture, June 1988.

[18] N. McKeown, “iSLIP: A Scheduling Algorithm for Input-Queued switches,” IEEE Transactions on Networking, Apr. 1999.

[19] K. L. E. Law and A. Leon-Garcia, “A large scalable ATM multicast switch,” IEEE J. Selected Areas Commun., vol15, no. 5, pp. 844-854, 1997.

[20] H. S. Kim, “Design and Performance of Multinet Switch: A Multistage ATM switch architecture with Partially Shared Buffers,” IEEE/ACM Trans. Networking, vol. 2, pp. 571-580, December 1994.

[21] J. N. Giacopelli, J. J. Hickey, W. S. Marcus, W. D. Sincoskie and M. Littlewood, “Sunshine: A High Performance Self-routing Broadband Packet Switch Architecture,” IEEE J. Selected Areas Commun. vol. 9, October 1991.

[22] N. McKeown, V. Anantharam, J. Walrand, “Achieving 100% Throughput in an Input-Queued Switch,” Proc. INFOCOM 96, pp. 296-302.

[23] M. J. Karol and M. G. Hluchyj, “The Knockout Switch: Principles and performance,” in Proc. 12th Conf. on Local Computer Networks, 1987, pp 16-22.

[24] K. Y. Eng, M. G. Hluchyj and Y. S. Yeh, “Multicast and Broadcast Services in a knockout Packet switch,” INFOCOM ’88, pp. 29-34, 1988.

[25] Rein J. F. de Vries, “ATM Multicast connections using the Gauss switch,” in Proc. GLOBECOM ’90, pp. 211-217.

[26] Mekkittikul, A and McKeown, N “A Starvation-free algorithm for Achieving 100% Throughput in an Input-Queued Switch,” Proc. of ICCCN ’96, October 1996, pp. 226 - 231.

[27] D. X. Chen and J. W. Mark, “Multicasting in SCOQ Switch,” INFOCOM ’94, pp. 290-297, 1994.

[28] H. J. Chao and B. S. Choe, “Design and Analysis of large-scale multicast output buffered ATM switch,” IEEE/ACM Trans. Networking, vol. 3, pp. 126-138, April 1995.

[29] R. Bakka and M. Dieudonne, “Switching circuit for digital packet switching network,” U. S. Patent 4 314 367, Feb. 2, 1982.

[30] S. Nojima et al., “Integrated services packet network using bus matrix switch,” IEEE J. Select. Areas Commun., vol. SAC-5, pp. 1284-1292, Oct. 1987.

[31] Y. Kato et al., “ Experimental broadband ATM switching system,” in Proc. GLOBECOM ’88, Hollywood, FL, Nov, 1988, pp. 1288-1292.

[32] J. Y. Hui and T. Renner., “Queueing Analysis for Multicast Packet Switch,” IEEE Transactions on Communications, vol. 42, no. 2/3/4, pp 723-731, Feb 1994.

[33] L. R. Goke and G. J. Lipovski, “Banyan networks for partitioning multiprocessor systems,” in Proc. 1st Annu. Int. Symp. Computer Architecture, Apr. 1979, pp. 168-177.

[34] J. H. Patel, “Processor-memory interconnections for multiprocessors,” in Proc. 6th Annu. Int. Symp. Computer Architecture, Apr 1979, pp. 168-177.

[35] K. E. Batcher, “Sorting networks and their applications,” in proc. Spring joint Comput. Conf., AFIPS, 1968, pp 307-314.

[36] A Huang and S Knauer, “Starlite: A wideband digital switch,” in Proc. GLOBECOM ’84, Atlanta, GA, Dec. 1984, pp 121-125.

[37] Duncan H. Lawrie, “Access and Alignment of Data in an Array Processor” IEEE Transactions on Computers, Dec. 1975, pp 1145-1155.

[38] Michael D. Schroeder et. al., “Autonet: A High-Speed, Self-Con figuring Local Area Network Using Point-to-Point links,” IEEE Journal of selected areas in communications. vol. 9. no.8, Oct. 1991.

[39] N. J. Boden et. al., “Myrinet: A Gigabit-per-second local area network,” IEEE Micro, pages 29-36, Feb. 1995.

[40] Ram Kesavan and Dhabaleswar K. Panda, “ Efficient multicast on irregular switch-based cut through networks with up-down routing,” IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 8, Aug. 2001.

[41] F.J. Alfaro, A. Bermudez, R Casado, F. J. Quiles, J.L Sanchez and J. Duato, “ On the Performance of Up*/Down* routing,”

[42] Mario Gerla, Prasanth Palnath, Simon Walton, Emilo Leonardi, Fabio Neri, “Multicasting in Myrinet - A High-Speed, wormhole-routing network,” IEEE 1996.

[43] Mario Gerla, Prasanth Palnath and Simon Walton, “Multicasting protocols for High-Speed, Wormhole-Routing Local Area Networks,” Technical Report 960010, Computer Science Dept., UCLA, February 1996.

[44] S. Owicki and A. R. karlin, “Factors in the performance of the AN1 computer Network,” Technical report 88, Digital Equipment Corporation Systems Research Centre, Palo Alto, CA, June 1992.

[45] Kees Verstoep, Koen Langendoen, Henri Bal, “ Efficient reliable multicast on Myrinet,” IEEE International conference on Parallel Processing, 1996.

[46] Ming-Huang Guo and Ruay-Shiung Chang, “Multicast ATM switches: Survey and Performance Evaluation,” ACM SICOMM Computer Communication Review, Volume 28, Issue 2, April 1998.

[47] Ramakanth Gunuganti, “Ω Switch, A high speed atm switch for multicast and unicast.” Masters thesis, University of Texas at San Antonio, Aug. 2000.

[48] W. E. leland, M. S. Taqqu, W. Willinger and D. V. Willson, “On the self-similar nature of Ethernet traffic (extended version),” IEEE/ACM Transactions on Networking, vol. 2, pp 1-15, Feb. 1994.

[49] V. Paxson and S. Floyd, “Wide area traffic: The failure of Poisson modeling,” IEEE/ACM Transactions on Networking, vol. 3, pp. 226-244, June 1995.

Vita

Rajesh Boppana was born in Andhra Pradesh, India on July 24, 1977, the son of Koteswara Rao and Rajyalakshmi. In 2000, he graduated from Jawaharlal Nehru Technological University, receiving the Bachelor of Technology with a major in Computer Science and Engineering. He entered the graduate program at UTSA in 2000.

He can be reached at $rajesh\_bin@yahoo.com$.